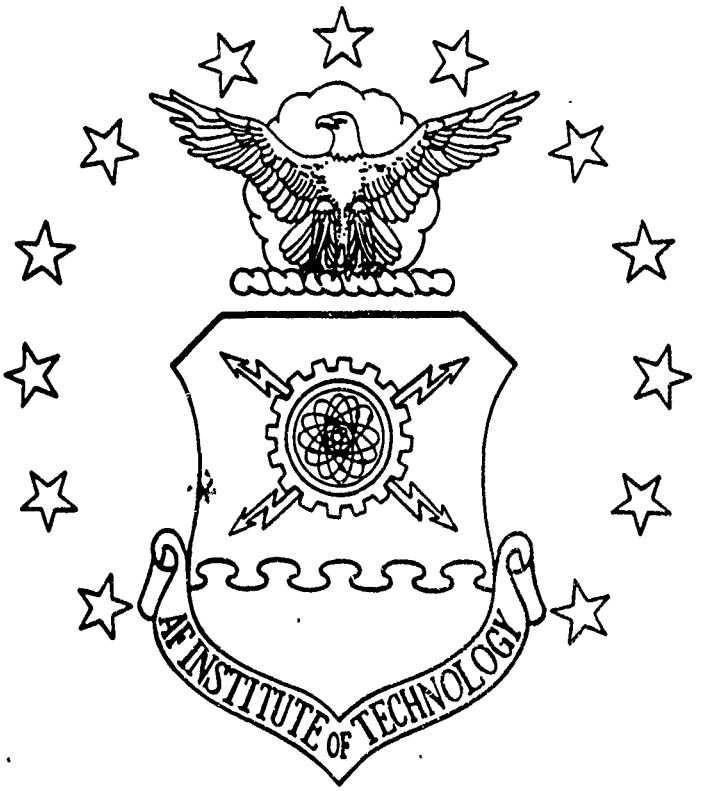


DEVELOPMENT OF
A FLIGHT INFORMATION SYSTEM
USING
THE STRUCTURED METHOD

THESIS

Yeong-Lae Kwak
Captain, ROKAF

AFIT/GCS/ENG/92M-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/92M-03

DEVELOPMENT OF A FLIGHT INFORMATION SYSTEM USING THE STRUCTURED METHOD

THESIS

Yeong-Lae Kwak
Captain, ROKAF

AFIT/GCS/ENG/92M-03

AFIT/GCS/ENG/92M-03

DEVELOPMENT OF A FLIGHT INFORMATION SYSTEM USING THE STRUCTURED METHOD

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Systems

Yeong-Lae Kwak, B.S.

Captain, ROKAF

March 1992

Preface

The purpose of this thesis was to develop a Flight Information System (FIS) using the Structured Method as a software development method and ORACLE as a database management system. The Korean Air Force was selected as a model. This thesis covered only the core part of the FIS. This thesis focused on two points: the application of software development tools and the application of the ORACLE products.

This thesis was accomplished through many people's encouragement and support. I would like to express my deep appreciation to those people who helped me during the last 21 months.

First of all, I would like to thank my thesis advisor Dr. Henry Potoczny and thesis committee members, Maj. Mark Roth and Maj. Paul Bailor. Their continuous guidance made it possible for me to complete my thesis.

I would also like to thank two USAF officers and an ORACLE contractor: Capt. Armin Sayson, Capt. Andrew Dymek, and David Roliff. They spent much of their time helping me. They helped my thesis work by checking grammar and solving miscellaneous problems.

Additionally, I would like to thank my family members and my wife's family members for their continuous support and encouragement from the other side of the globe.

Finally, I would like to thank my wife Yangsook Lee for sharing all my difficulties, giving her devoted support, and for enduring a long lonely time.

Yeong-Lae Kwak

Table of Contents

List of Figures

List of Tables

Abstract

This thesis documents the development of a database system for the *Flight Information System* (FIS) of the Korean Air Force. The scope of the FIS is too large to be covered by this thesis. Thus, this thesis covers only the core part of the FIS due to the limitation of time and manpower. The users of the FIS can be grouped into five categories: Combat Air Command (CAC), wing, squadron, control tower, and other departments. The scope of the FIS can be divided into six categories: organization, pilot, aircraft, flight mission, flight order, and flight sortie.

This thesis uses the structured method. Structured analysis and structured design techniques are mainly used two techniques. Many tools such as Data Flow Diagram (DFD), Data Dictionary (DD), Process Specification, Entity Relationship Diagram (ERD), and Structure Chart are used. ORACLE was used as a database management system. SQL, SQL*Forms, SQL*Menu, and Pro*Ada are ORACLE products used in this thesis. This thesis was developed by following three steps.

The first step was the *Analysis*. The user required data were identified using DFD, DD, and Process Specification. Then, the perception of the real world of the FIS was modeled into a database structure using ERD.

The second step was the *Design*. Database tables were generated from the ERD and the Structure Chart with Module Description were generated.

The third step was the *Implementation and Test*. Each module of the structure chart were implemented and tested using top-down development method. Pro*Ada and SQL*Forms were used as programming tools.

This thesis focused not only the development of the FIS but also the application of the software development method, the structured method, and its tools such as DFD, DD, ERD, and so on. Also, the use of ORACLE was a important part of this thesis too.

Development of

a Flight Information System

using

the Structured Method

I. Introduction

1.1 Background

In 1985, the Republic of Korean Air Force (ROKAF) developed a database system called the Flight Information System (FIS). The purpose of this system was to gather information related to air operations and to support the policy decisions of the Air Force. The FIS included information related to pilots, aircraft, flight-time, flight-sorties, flight missions, and others. It was a big system with more than 200 programs (approximately 50,000 lines of code). However, the FIS was not a successful system. It contained many structural problems and program errors. The system failed for a number of reasons. The cause of software failure was due to the designer's lack of software development experience using database systems, insufficient understanding of database systems, and incorrect use of software development tools.

At this time, the FIS does not support the users sufficiently and contains many incorrect data. Due to the lack of computer assistance, much of the data management tasks are done manually. Therefore, much time is spent gathering, processing, and extracting required information. However, due to human errors and inconsistent extraction of the required information by different people, the reliability of the extracted information is questionable.

1.2 User Requirements

The users of the FIS require a reliable and convenient new FIS. The following five groups are the users of the FIS and will become the users of the new FIS which will be developed in this thesis:

- Combat Air Command (CAC),
- Flight Wing,
- Flight Squadron,
- Other Departments,
- Control Tower.

The CAC is the top department of the Air Force related to the air operations. It issues regular and immediate flight-orders to each wing. It also watches the current flight status of each wing and analyzes the flight result data such as flight-time, flight-sortie, and flight-missions. The CAC also orders a pilot to move to certain organizations.

A wing receives flight-orders from the CAC at any time. Like the CAC, each wing writes flight-orders and dispatches them to each of their squadrons to accomplish the flight-orders received from the CAC. Additionally, each of the wing sends flight-orders for the squadrons to perform the missions which are planned by itself and to upgrade the wing's mission capability.

A flight squadron is an organization which flies with its own aircraft. It receives flight-orders from the wing at any time. Everyday, it writes flight-plans and executes them. Flight plans change continuously until take-off because of new flight-orders and bad weather conditions. Each flight squadron has several types of aircraft and numerous pilots. The pilots of each squadron fly often to perform their squadron's flight-plans.

Other departments are not related to flight directly but need information related to the flight. Other departments include Headquarter, Academy, Logistics Command, Department of Defense and others. The pilots of other departments fly regularly to maintain their own flight capability.

A control-tower is an organization which controls and records the exact take-off and landing time of each aircraft to prevent collisions. Each air-base has a control-tower to support aircrafts which uses the air-base's runway. Sometimes aircraft land at unscheduled bases. Therefore, every control-tower needs continuously updated current flight plans and statuses of all the Air Force.

A new FIS development team was organized and surveyed the requirements of the users. The category of the new FIS is too large and the users require much detail. However, due to the limitation of time and manpower, this thesis tries to develop only the core part of the new FIS.

The core part of the new FIS manages important data of the FIS and require complex program application. It consists of three categories and each of them are discussed below.

The first category of requirements is information on pilots and aircraft. The data on each pilot and aircraft changes continuously. All the users, with the exception of the control-tower, require correct and detailed current information on each pilot and aircraft.

The second category is the flight-order transfer system. There are two kinds of flight orders: cac-order and wing-order. Cac-orders are sent from CAC to each wing continuously. And wing-orders are sent from each wing to their squadrons continuously. The purpose of flight-orders can be grouped into three. The first is to confront emergency situations. By sending flight-orders with a specific flight mission, the Air Force can confront an emergency situation. The second is to initiate regular and special military operations. The CAC or each wing may plan air operations for various reasons. The air operations are initiated by sending flight-orders with flight mission. And the third is to increase or to maintain the flight capability of wings or squadrons. The CAC, each wing, and each squadron require a reliable, fast flight-order transfer system.

The last category of user requirements is flight status display system. Each squadron and several other departments plan and conduct flight continuously. All the organizations must exchange flight status information which includes flight planning and current flying information. By sharing flight status information, all organizations have the flight status information available on screens which show how many sorties are ready to fly, how many aircraft are in the air, and how many flights were executed that day. More detailed information can be known with such a system.

1.3 Approach and Methodology

The new FIS is a data processing oriented system. Almost all information such as pilots, aircraft, missions, and flight-sorties can be stored in a database system and displayed or printed easily for the users. However, other information, such as flight order and flight status, require complex software application techniques. Such information must be processed and displayed simultaneously as the users input them.

The Korean Institute of Defense Analysis (KIDA) recommends the structured method, discussed in section 2.2, as the Korean military software development method. The ORACLE Re-

lational Database Management System (RDBMS), discussed in detail in section 2.3, is one of the most widely known Data Base Management System (DBMS) and is expected to be used more widely than others.

In this thesis, the VAX 8650[1] with the Virtual Memory System (VMS) operating system will be used in the development of the FIS. The structured method, a software development method, using ORACLE Relational Database Management System (RDBMS) Version 6.0 will be used to develop the FIS. Almost all programs will be developed using SQL*Forms. However, some programs which require complex procedures and/or concurrent processing, will be developed with Pro*Ada which will also be discussed in section 2.3.

Following are the selected software development steps which will be applied in this thesis. The analysis phase begins by identifying the required data of the users. Data Flow Diagrams (DFDs), Data Dictionary (DD) and process specifications will be used as analysis tools. The next step is modelling the database. Entity-Relationship Diagrams (ERDs) are to be used in this step. In the design phase, database tables will be generated from the ERDs and DD using relational scheme normalization technique which will be discussed in section 2.4. Then, a structure chart will be derived from the DFD which was derived previously in the analysis phase. What happens inside each module of the structure chart will be described in detail using module description to aid the implementation of it. The final phase is the implementation and testing of each module.

1.4 Sequence of Presentation

Chapter 2 briefly reviews the required knowledge to understand this thesis and the rules used in this thesis. The structured method with its tools such as DFD, DD, ERD and Structured Chart with their notations are discussed briefly. Also, the relational scheme normalization technique will be discussed. Chapter 3 describes the structured analysis process used in the development of the FIS. The ERD, DFD and DD are used as structured analysis tools. Chapter 4 describes the process of structured design of the FIS. Database design and structured charts will be discussed. Chapter 5 describes the process of implementation and testing of the FIS system. Finally, conclusions and recommendations are presented in Chapter 6.

[1]VAX 8650 is made by Digital Equipment Corporation (DEC) and has 48 Megabytes of main memory.

II. Literature Review

2.1 Introduction

This chapter provides the basic knowledge required to write or to understand this thesis and the notations used in this thesis. The basic knowledge can be divided into three major categories: structured method, ORACLE, and normalization technique. The structured method is a software development methodology which is appropriate for a data processing system. ORACLE is one of the most widely known relational database management systems. Normalization techniques are an important database design technique to keep the relational scheme in high integrity and maintainability.

2.2 Structured Method

The causes of failure of conventional electronic data processing (EDP) systems can be found from the following (5):

- Poor system analysis,
- Little or no control over design and code,
- Bottom-up development and integration.

Structured techniques are new software development techniques opposite to conventional techniques. One of the first structured techniques developed is structured programming, and was introduced in 1972-73. Structured design and analysis are two other structured techniques introduced in 1974-75 and 1976-77, respectively. A fourth structured technique is top-down development. Following are discussions on structured analysis, structured design, and the tools used in those techniques.

2.2.1 Structured Analysis. Structured analysis is one of the most important structured techniques. In classical analysis, the analyst prepares a document describing the proposed system. However, the document requires too much time to understand and even makes it impossible to be understood by the users. The reasons for the users' difficulty with classical functional specifications can be summarized as follows (5):

- Classical functional specifications are monolithic,
- Classical functional specifications are redundant,
- Classical functional specifications are difficult to modify and maintain,
- Classical functional specifications make too many assumptions about implementation details.

Structured analysis introduces a new and different kind of functional specification called structured specification which uses graphical documentation tools. Graphical tools make the specifications easily understood by the users. Data flow diagrams, data dictionaries, entity relationship diagrams, state transition diagrams and structured English are the most commonly used tools.

2.2.1.1 Data Flow Diagrams. One of the key goals of structured analysis is to partition the area to be specified. Then, an integrated set of process specifications can be written rather than a monolithic one. DeMarco defines a DFD as (24)

> A *Data Flow Diagram* is a network representation of a system. The system may be automated, manual or mixed. The data flow diagram portraits the system in terms of its component pieces, with all interfaces among the components indicated.

DFDs used in this thesis are made up of the following five items:

- Terminators, represented by named boxes,
- Processes, represented by circles,
- Data storages, represented by long ovals,
- Data flows, represented by named arrows with solid lines,
- Control flows, represented by named arrows with dotted lines.

A terminator is a person or organization lying outside the context of a system that is a net originator or receiver of system data. A process is a transformation of incoming data flow(s) into outgoing data flow(s). A data storage is a temporary or permanent data repository. A data flow is a pipeline through which one or more pieces of information flow. A control flow is a pipeline through which one or more control signals flow.

The first step in drawing a DFD starts with a drawing of a context diagram which shows net input and output data flows between terminators and a process which includes a whole system. Then, the system must be broken down into several processes. Considering the capacity of the human brain, DeMarco says breaking the system into seven or fewer pieces is the best partition for managing them efficiently (24). If a decomposed process is too big to be a functional primitive, break it down again until all processes become functional primitives. A functional primitive is a process which can't or need not be decomposed any more.

A decomposed DFD does not show the source or destination of the data which flows from/to outside of the domain. If a decomposed DFD has many data flows from/to outside of the domain, it is difficult to figure out the source or destination of them. In this thesis, the source or destination of data, which flows between decomposed DFD and outside of the decomposed DFD, are described as in Figure 1.

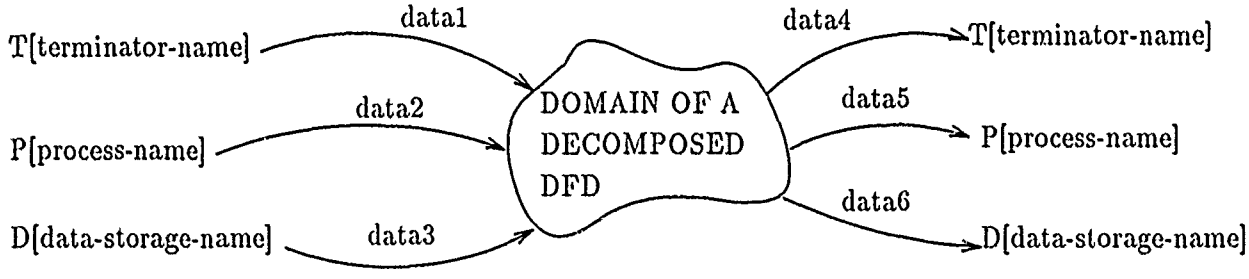


Figure 1. Notation of Data Source & Destination of Decomposed DFD

2.2.1.2 Data Dictionaries. The DD is an important part of the structured specification. Without DDs, DFDs are just pictures that show some idea of a system. DFDs and the DD have to be considered together. The DD defines data flows, components of data flows, files, and processes. Yourdon describes a DD as follows (6):

> The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and systems analyst will have a common understanding of all inputs, outputs, components of stores and intermediate calculations.

There are many common notational schemes used by system analysts. Table 1 shows the notations which will be used in this thesis.

NOTATION	MEANING
=	is composed of
+	and
( )	enclosed component is optional
{ }	enclosed component is iterative
[ ]	select one of the options enclosed in the brackets
* *	comment
:	separates alternative choices in the [ ] construct

Table 1. Notation of Data Dictionary

2.2.1.3 Process Specifications. A process specification describes what happens inside the each primitive process in a DFD using structured English, pre-post conditions and decision tables. It should state what has to be accomplished by the process rather than how the process should accomplish it. Each primitive process must be described by a process specification precisely but clearly.

2.2.1.4 Entity Relationship Diagrams. An ERD is a type of database modelling tool which was first introduced in 1976. Because of its simplicity, readability, and because it is easy to learn, it is known widely and applied to a variety of industrial and business applications. The basic ERD consists of three classes of objects: entities, relationships, and attributes. Figure 2 shows the notation of these three classes.

Entities are the principal data objects on which information is to be collected; they usually denote a person, place, thing, or event of informational interest (23). A weak entity, on the other hand, is different from an entity. The existence of a weak entity is dependent on the existence of a strong entity. A weak entity set, in an ERD, is represented by a doubly outlined box and is connected to a strong entity set by arrowed lines. For example, in Figure 2, the existence of a weak entity set *dependent*, depends on the existence of a strong entity set *officer*.

Relationships represent the real-world associations among one or more entities and, as such, have no physical or conceptual existence other than that which is inherited from their entity associations (23). Relationships are described in terms of degree and connectivity as shown in Figure 3.

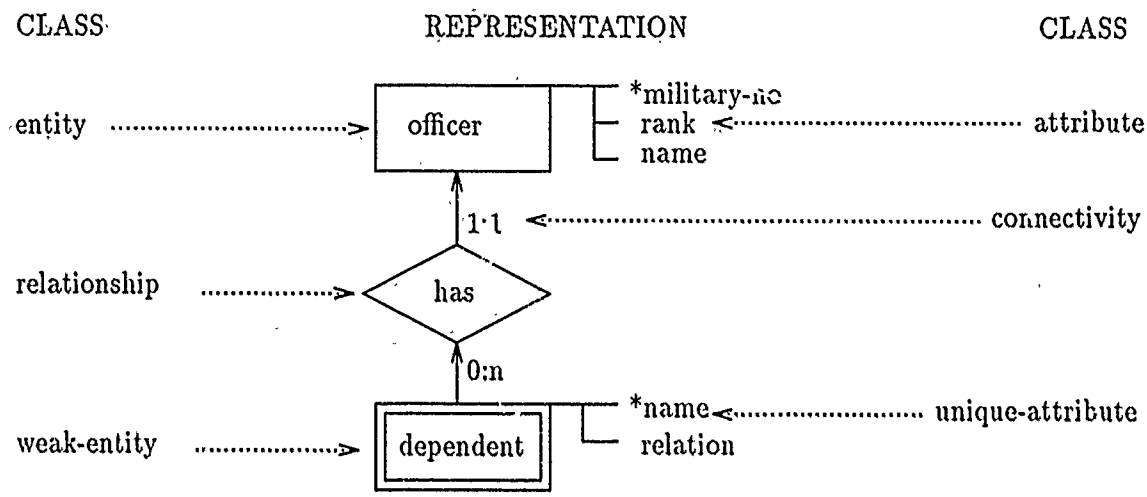


Figure 2. Notation of ERD

The degree of a relationship is the number of entities associated in the relationship. The connectivity of a relationship describes the mapping of the associated entity occurrences in the relationship.

Attributes are characteristics of entities or relationships that provide descriptive detail about them (23). A particular occurrence of an attribute within an entity or relationship is called an attribute value.

2.2.2 Structured Design. The term structured design was introduced by IBM in the IBM Systems Journal in 1974 (4). Structured design is not the same as top-down design. Structured design is a collection of guidelines and techniques to help the designer distinguish between good design and bad design at the modular level (4). The structure chart is a tool of structured design.

A structure chart is a graphical technique for documenting the overall architecture of a large program or system (4). Structure charts do not show all the detailed decisions and loops inside the module but show all input and output data between other modules.

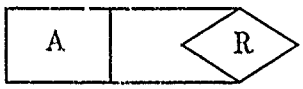
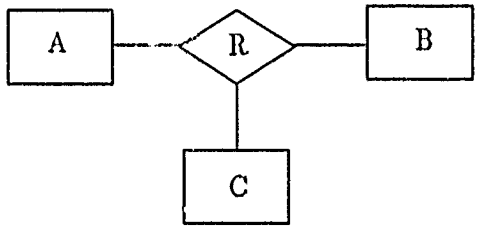
CONCEPT	REPRESENTATION	MEANING
DEGREE - unary		-Entity A has relationship R with entity A itself
- binary		-There is relationship R between entity A and entity B
- ternary		-There is relationship R among entity A, entity B, and entity C
CONNECTIVITY		-Entity A has minimum y, maximum z relationship to entity B -Entity B has minimum w, maximum x relationship to entity A

Figure 3. Degree & Connectivity of ERD

2.3 ORACLE

ORACLE ia a relational Database Management S\ .tem (DBMS) developed by the ORACLE Corporation[1]. The first version of ORACLE was developed in 1979 and installed on a DEC PDP-11 computer system. Oracle evol:ed into a 4GL product that can be run on a variety of mainframes, mini computers, and personal computers. It supports a large number of operating systems such as MS-DOS, UNIX, VM/SP, MVS/SP, MVX/XA, and VMS (25). The structure of the ORACLE system is shown in Figure 4.

[1]ORACLE Corporation, formerly Relational Software Inc., was formed in 1977 and is located in Belmont, CA.

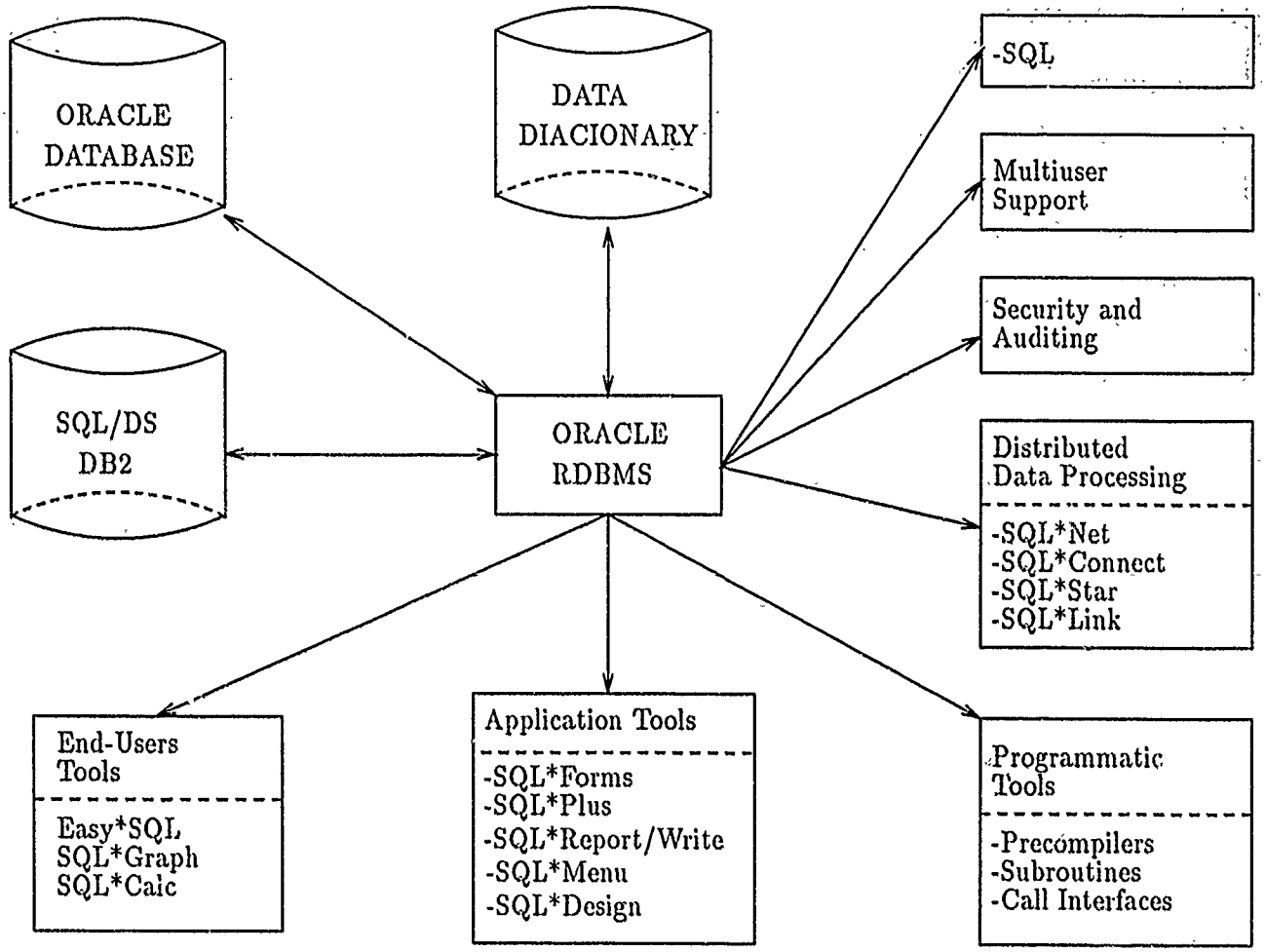


Figure 4. Oracle Facilities

2.3.1 ORACLE RDBMS. The ORACLE RDBMS is the central ORACLE product. It includes the database manager and several tools intended to assist users and Database Administrators (DBAs). The core of the RDBMS is the kernel which handles the following tasks:

- manages storage and destination of data,
- controls and limits data access and concurrency,
- allows backup and recovery of data,
- interprets SQL and Procedural SQL (PL/SQL).

2.3.2 SQL. SQL is an English-like language that is used for most database activities. SQL was developed and defined by IBM Research, and has been refined by the American National Standards Institute (ANSI) as the standard language for relational database management systems. SQL language statements can be divided into four categories (10):

- Queries: Queries are statements that retrieve data, in any combination, expression, or order. Queries usually begin with the SQL reserved word SELECT,
- Data Manipulation statements (DML): DML statements are used to change data by inserting, updating, or deleting. DML statements include INSERT, UPDATE, DELETE, LOCK, COMMIT WORK, and ROLLBACK WORK statements,
- Data Definition statements (DDL): DDL statements are used to define and maintain database objects and database tables. DDL statements include CREATE, ALTER, and DROP statements,
- Data Control statements (DCL): DCL statements allow one user to let another users privileges. DCL statements include GRANT and AUDIT statements.

*2.3.3 SQL*Menu.* SQL*Menu is one of ORACLE products which allows the creation of customized menu systems. It provides uniform access to all ORACLE software products such as the ORACLE RDBMS, SQL*Forms, SQL*Plus, and SQL*Graph (19). Also, it is used to access any software product which runs on the computer operating system. Without any programming, it provides high security.

*2.3.4 SQL*Forms.* SQL*Forms is one of Oracle products which lets a user to interact with his/her database through screen forms. With SQL*Forms, the user can design a form on the screen and use it to access and read or update data inside the ORACLE database. SQL*Forms provides the ability to

- insert data into the database by typing the data directly into the fields,
- view, update, or delete several records on the screen at one time,
- type query conditions directly into the fields you want to query.

*2.3.4.1 SQL*Forms Components.* The components of SQL*Forms include:

- SQL*Forms(also called IAD), the Interactive Application Designer, which creates or modifies the form in the database. It is the main component which can call all the others. It is also executed when you choose CREATE or MODIFY from the CHOOSE FORM window,
- IAC, the Interactive Application Converter, which converts a form between database and INP format. It is executed when you select GENERATE or LOAD from the CHOOSE FORM window,
- IAG, the Interactive Application Generator, which reads an INP file and generates a FRM file. It is executed after IAC when you select GENERATE from the CHOOSE FORM window,
- IAP(also called RUNFORM), the Interactive Application Processor, which reads a form from a FRM file and runs it. It is executed when you choose RUN from the CHOOSE FORM windows.

2.3.4.2 Triggers. A trigger is a set of commands that are started by a certain event when a form is being run. Each trigger may be composed of one or more steps, each of which contains one command. Triggers can:

- validate data entry several ways,
- protect the database from operator errors such as the entry of duplicate records, or the deletion of vital records,
- limit operator access to specified forms,
- display related field data by performing table look-ups,
- compare values between fields in the form,
- calculate field values and display the results of field calculations in different fields,
- enforce block coordination during insert, update, or query operations,
- expand the functionality of function keys,
- perform complex transactions, such as verifying batch totals.

Three kinds of commands can be used in triggers:

- SQL commands, such as

```
SELECT name
INTO   :pilot.name
FROM   pilot
WHERE  pilot-id = :pilot.pilot-id;
```

- SQL*Forms commands, such as

```
#EXECMACRO GOBLK pilot; EXEQRY;
```

- User exits which call user programs written in a programming language such as C or Ada.

Triggers are leveled into three (FORM, BLOCK, and FIELD) and each of them can be associated with five kinds of events (Entry, Query, Change, Exit, and Key-strokes). User-named trigger is a another kind of trigger which can be used from other triggers. Figure 5 shows details of that (14).

*2.3.5 Pro*Ada.* The data associated with an ORACLE RDBMS can be accessed and manipulated by application programs written in COBOL, FORTRAN, C, PL/I, Pascal, Ada, and assembly language. The Pro*Ada precompiler, an application programming tool, allows an Ada program to utilize embedded SQL language constructs (12). A Pro*Ada source program contains both SQL and Ada language constructs. The Pro*Ada precompiler translates each SQL statements in the program into ORACLE runtime calls.

Using Pro*Ada can take both advantage features of Ada and SQL. Ada provides the procedural language support needed for the application, while the embedded SQL statements provide direct access to ORACLE along with the data manipulation functionality of SQL. Thus, it is possible to develop more powerful and more flexible programs.

A Pro*Ada program can be developed through the following steps:

TRIGGER EVENT	LEVEL			
		BLOCK		
	FIELD	RECORD	BLOCK	FORM
Entry	Pre-Field	Pre-Record	Pre-Block	Pre-Form
Query		Post-Query	Pre-Query	
Change	Post-Change	Pre-Delete Post-Delete Pre-Insert Pre-Update Post-Update		
Exit	Post-Field	Post-Record	Post-Block	Post-Form
Key-strokes	Key			
User-named	User-named			

Figure 5. Trigger Types

1. Define the abstract problem, including the selection of algorithms, etc,
2. Design the software, including package and main procedure specification. Code the specifications first and then the bodies,
3. Precompile the Pro*Ada program (filename.pad), resulting in Pro*Ada output files (filename.ora_dcl and filename.ada),
4. Compile the Pro*Ada output files, resulting in compilation units which are added to the Ada program library,
5. Link the object modules with the required ORACLE and Ada runtime libraries,
6. Run the program.

2.4 Normalization

In general, the goal of a relational database design is to generate a set of relation schemes that allow storing information without unnecessary redundancy, yet allow easy retrieval. A badly designed relational database scheme may have the following undesirable properties (7):

- Repetition of information,
- Inability to represent certain information,
- Loss of information.

Normalization is a process to generate relation schemes in Normal Form (NF) which make it possible to maintain relation schemes in high integrity and maintainability.

2.4.1 First Normal Form (1NF). A relation scheme is 1NF if all underlying domains contain only atomic values, that is, there are no repeating groups (domains) within a tuple. The advantage of 1NF over unnormalized relations is its simplicity and the ease with which one can develop a query language for it. The disadvantage is the requirement for duplicate data.

2.4.2 Second Normal Form (2NF). A relation scheme is in 2NF if it is in 1NF and every nonkey attributes is fully dependent on the primary key. This means that any Functional Depen dency (FD) within the relation must contain all components of the primary key as the determinant, either directly or transitively.

2.4.3 Third Normal Form (3NF). A relation scheme R is in 3NF if for all functional dependencies that hold on R of the form $X \longrightarrow A$, where $X \subseteq R$ and $A \in R$, at least one of the following conditions holds (7):

- X is a superkey for scheme R,
- A is a member of a candidate key for scheme R,
- $X \longrightarrow A$ is a trivial functional dependency. (that is, $A \in X$)

2.4.4 Boyce-Codd Normal Form (BCNF). BCNF is a stronger form of normalization than 3NF. BCNF eliminates the second condition for 3NF, which allowed the right side of the FD to be a member of a candidate key. A relation scheme R is in BCNF if for all functional dependencies that hold on R of the form $X \longrightarrow A$, where $X \subseteq R$ and $A \in R$, at least one of the following conditions holds:

- X is a superkey for scheme R,
- $X \longrightarrow A$ is a trivial functional dependency (that is, $A \in X$).

2.5 Summary.

This chapter presented the basic knowledge required to understand this thesis and the notations used in this thesis. The structured method of software development was described first. The structured analysis technique and the tools used in the technique such as DFDs, DDs, process specifications and DRDs were described. The structured design technique with the structure chart, a structured design tool, were discussed. Second, the ORACLE RDBMS and its several products were described briefly. Finally, the relational scheme normalization technique was described.

III. Analysis of User Requirements

3.1 Introduction

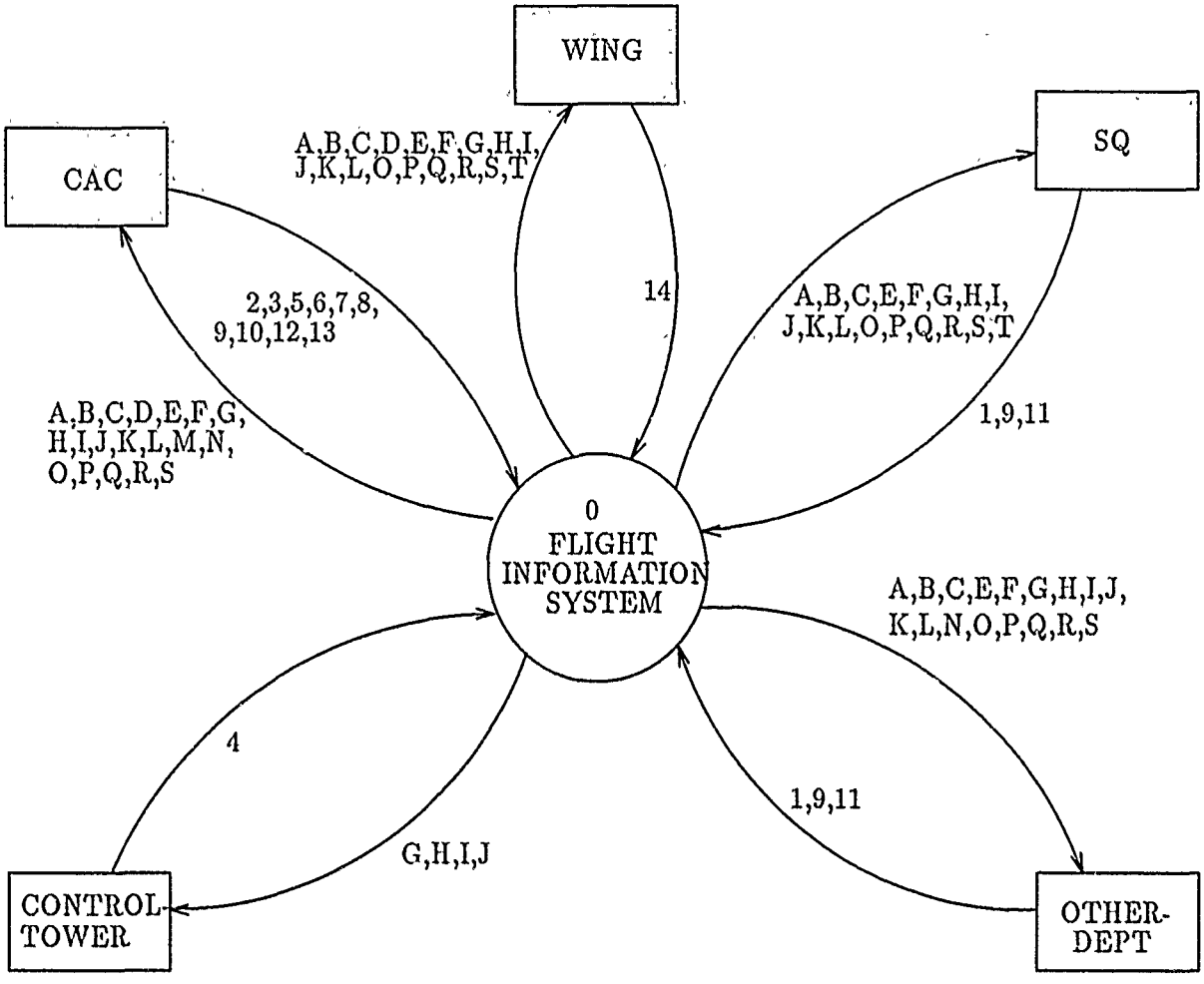
This chapter analyzes the FIS using the structured analysis technique which was discussed in Section 2.2.1. The analysis phase of the FIS is divided into two major steps. The first step is identifying the user required data using the structured analysis tools such as the DFD, DD, and process specification. The next step is database modelling which generates the structure of a database from the perception of the real FIS. ERD will be used as a database modelling tool.

3.2 Identify User Required Data

The top-down approach will be used to identify user-required data. The first step is to derive context diagram which shows all terminators, input data, and output data of the FIS. The FIS will be decomposed into several processes until each process becomes a functional primitive. The definition of all data, represented on the DFD, will be described in detail in the DD which will be placed in Appendix A. The functions of each functional primitives will be described by process specifications.

3.2.1 Context Diagram. Figure 6 is the context diagram of the FIS. This context diagram consists of one processor, five terminators, fourteen input data, and twenty output data. It shows all user groups around the FIS and all data flows between each user and the FIS. Table 2 shows the source(s) or destination(s) of each input and output data. It shows that almost of all data are shared by different users. Each input data with the exception of ac-rec, pilot-rec, and sortie-rec comes from one terminator, but each output data with the exception of org all, after processed by the FIS, is shared by two or more user groups.

3.2.2 Partition of the FIS. The context diagram was partitioned into six major groups depending on the characteristics and functions of each. Figure 7 shows the partitioned DFD of the FIS. It consists of six processes, nine data-storages, and many data flows. It does not show the terminators around the system. However, one end of each arrow which connects between the Figure 7 and the outside of it shows the source or destination of the data on the arrow. Functions of each processes are:



INPUT DATA

1. ac-rec
2. ac-type
3. cac-order-rec
4. exec-time
5. exercise
6. grand-msn
7. msn-rec
8. other-dept
9. pilot-rec
10. rank
11. sortie-plan
12. sq-rec
13. wing-rec
14. wing-order-rec

OUTPUT DATA

A. ac-rec
B. ac-tot-status
C. ac-type
D. cac-order-rec
E. exercise
F. flight-record
G. flight-status-all
H. flight-status-list
I. flight-status-org
J. flight-status-wing
K. grand-msn
L. msn-rec
M. org-all
N. other-dept
O. pilot-list1
P. pilot-list2
Q. rank
R. sq-list
S. wing-list
T. wing-order-rec

Figure 6. Context Diagram

	NO	DATA-NAME	SOURCE/DESTINATION				
			CAC	WING	SQ	OTHER-DEPT	CONTROL-TOWER
	1	ac-rec			√	√	
	2	ac-type	√				
	3	cac-order-rec	√				
	4	exec-time					√
I	5	exercise	√				
N	6	grand-msn	√				
P	7	msn-rec	√				
U	8	other-dept	√				
T	9	pilot-rec	√	√	√	√	
	10	rank	√				
	11	sortie-plan	√			√	
	12	sq-rec	√				
	13	wing-rec	√				
	14	wing-order-rec		√			
	1	ac-rec	√	√	√	√	
	2	ac-type	√	√	√	√	
	3	cac-order-rec	√	√	√	√	
	4	cac-order-rec	√	√			
	5	exercise	√	√	√	√	
	6	flight-record	√	√	√	√	
	7	flight-status-all	√	√	√	√	√
O	8	flight-status-org	√	√	√	√	√
U	9	flight-status-list	√	√	√	√	√
T	10	flight-status-wing	√	√	√	√	√
P	11	grand-msn	√	√	√	√	
U	12	msn-rec	√	√	√	√	
T	13	org-all	√				
	14	other-dept	√			√	
	15	pilot-list1	√	√	√	√	
	16	pilot-list2	√	√	√	√	
	17	rank	√	√	√	√	
	18	sq-list	√	√	√	√	
	19	wing-list	√	√	√	√	
	20	wing-order-rec		√	√		

Table 2. Data Source and Destination

- **Process 1** Input and output all information related to organization,
- **Process 2** Input and output all information related to pilot,
- **Process 3** Input and output all information related to aircraft,
- **Process 4** Input and output all information related to flight mission,
- **Process 5** Input and output all information related to flight-order,
- **Process 6** Input and output all information related to flight-sortie.

3.2.3 Partition of Organization. Figure 8 shows the decomposed DFD from Process 1. Organization consists of four groups: *CAC, wing, squadron,* and *other-department. CAC* commands all *wings* and each *wing* commands their *squadrons*. Each *wing* is committed to a *grand-msn* such as flight, carry, observe, or train. *Other-departments* such as the headquarter, the logistic command, and the Air Force academy are organizations which are not directly related to air operation. Since the FIS focuses to *wing* and *squadron*, the size of an *other-department* is much bigger than that of a *wing* or a *squadron*. For example, the headquarters of the Air Force are a unit organization like a *squadron* is.

Sometimes *wings* and *squadrons* may be created or deactivated. *CAC* inputs all changed information related to organizations and current information related to organizations are shared by all organizations with the exception of *control-tower*. The following are process specifications of each functional primitive shown in Figure 8:

Process 1.1

```
begin
repeat until user exit the program
    accept wing-code;
    if the wing-code exists in wing table
        then display wing-rec;
    end if;
    accept wing-rec;
    check validity of grand-msn-code;
    if wing-code exists in wing table
        then update wing-rec;
        else insert wing-rec;
```

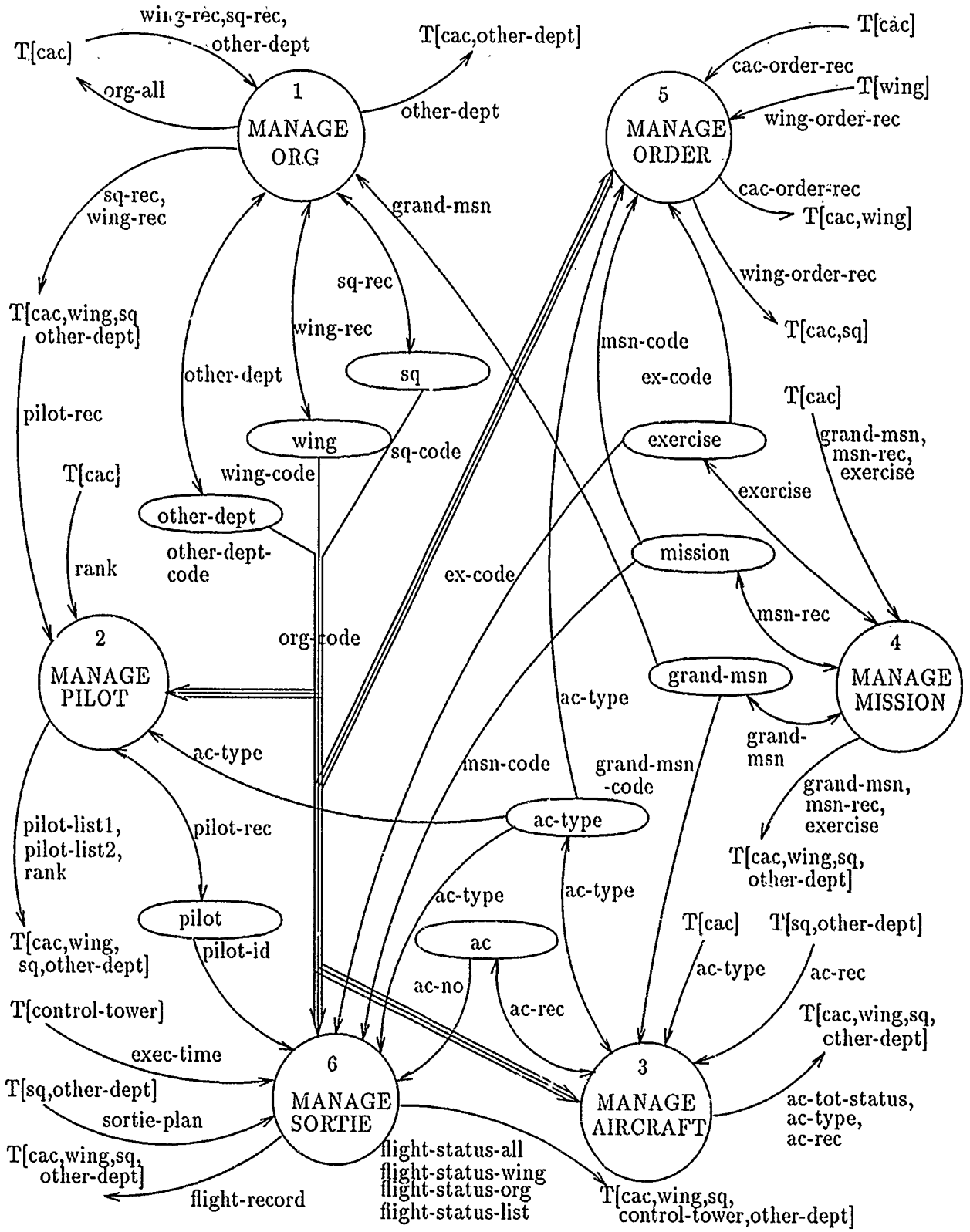


Figure 7. Overview DFD of the FIS

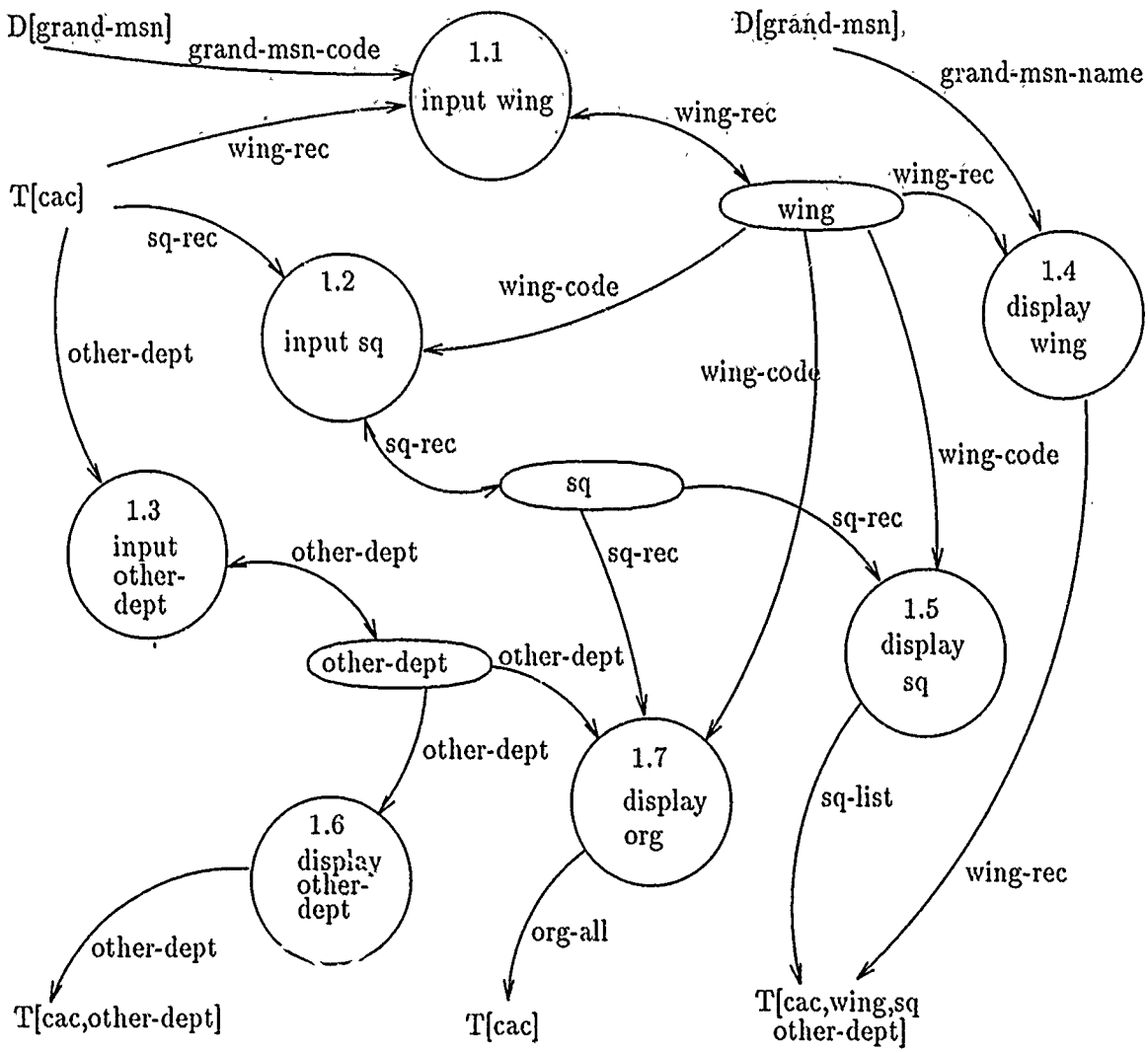


Figure 8. DFD of Organization

```
            end if;
        end repeat;
        end;
Process 1.2
        begin
        repeat until user exit the program
            accept sq-code;
            if sq-code exists in sq table
                then display sq-rec;
            end if;
            accept sq-rec;
            check validity of wing-code;
            if sq-code exists in sq table
                then update sq-rec;
                else insert sq-rec;
            end if;
        end repeat;
        end;
Process 1.3
        begin
        repeat until user exit the program
            accept other-dept-code;
            if other-dept-code exists in other-dept table
                then display other-dept;
            end if;
            accept other-dept;
            if other-dept-code exists in other-dept table
                then update other-dept;
                else insert other-dept;
            end if;
        end repeat;
        end;
Process 1.4
        begin
        display all wing-rec from wing order by grand-msn-code, wing-code;
        end;
Process 1.5
        begin
        display all sq-rec from sq order by wing-code, sq-code;
        end;
Process 1.6
        begin
        display all other-dept from other-dept order by other-dept-code;
        end;
Process 1.7
```

```
begin
display 'cac' on the screen;
repeat until data end
      display next record of wing table;
      display all sq-rec of the wing from sq order by sq-code;
end repeat;
display all other-dept from other-dept order by other-dept-code;
end;
```

3.2.4 Partition of Pilot. A *pilot* is a person who is committed to control a specific aircraft type to perform the flight mission of the Air Force. *Pilot* includes all persons who are enrolled to any pilot training course of the Air Force whether he is a student or an instructor. Each *pilot* has a unique *pilot-id, name, class, blood-type, pilot-date, job,* and *pilot-status.* All *pilots* are qualified to control a specific aircraft type. Occasionally, a *pilot* moves from one organization to another. Squadron must continuously check whether a *pilot* is in *ready-to-take-off*, on temporarily *duty-off*, *in-hospital*, or *in-vacation* to estimate the *ready-mission* status.

Figure 9 shows the decomposed DFD from the Process 2. *Pilots* are recorded in the FIS when they enter a pilot training course of the Air Force and deleted when they retire or leave the Air Force. Each organization must update information of the *pilot* continuously to make it possible for users to read the newest information at any time. The following are process specifications of each functional primitive of Figure 9:

Process 2.1

```
begin
repeat until user exit the program
      accept rank-code;
      if the rank-code exists in rank table
            then display rank;
      end if;
      accept rank;
      if rank-code exists in rank table
            then update rank;
            else insert rank;
      end if;
end repeat;
end;
```

Process 2.2

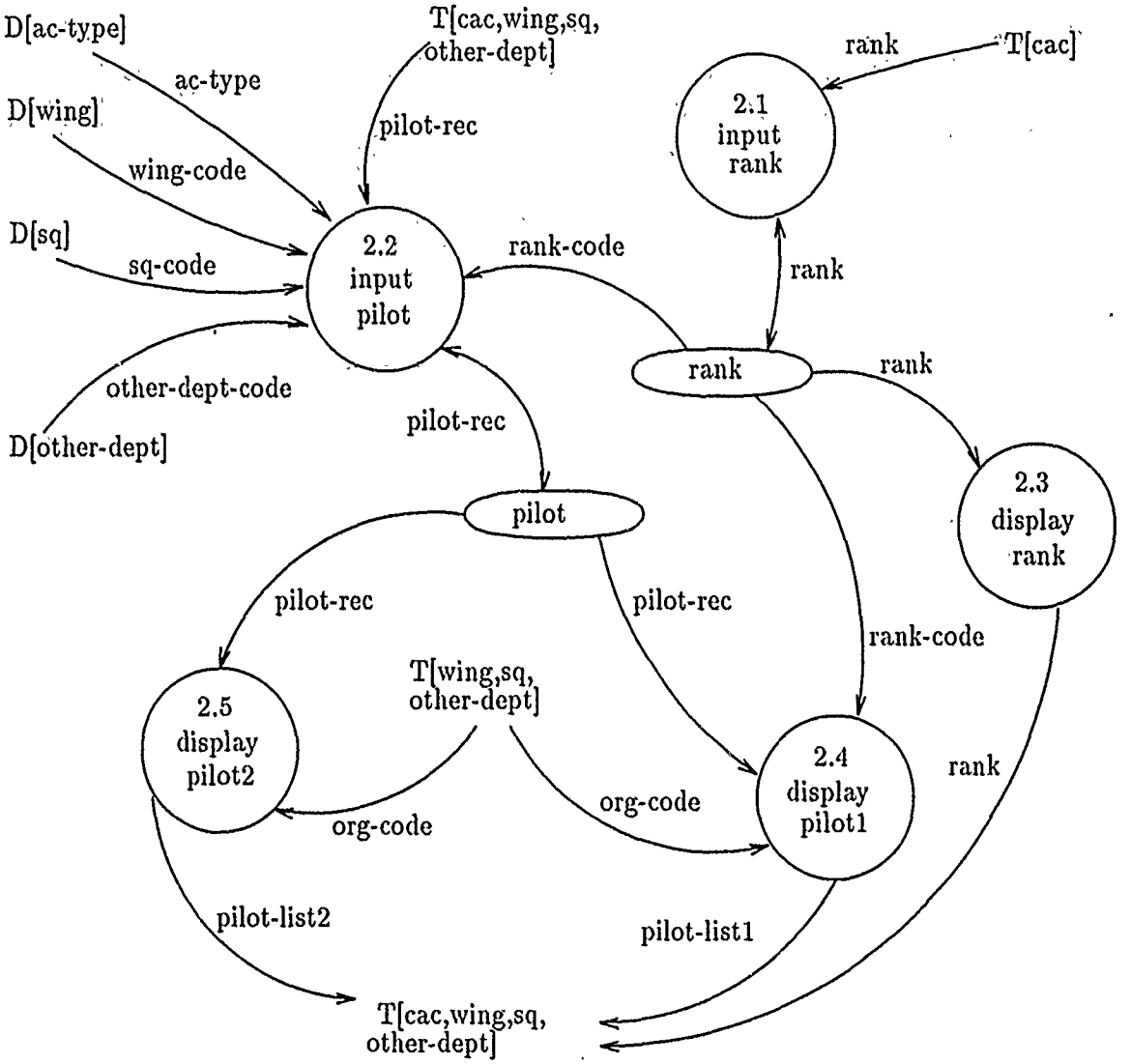


Figure 9. DFD of Pilot

```
        begin
        repeat until user exit the program
                accept pilot-code;
                if pilot-code exists in pilot table
                        then display pilot-rec;
                end if;
                accept pilot-rec;
                check validity of ac-type and org-code;
                if pilot-code exists in pilot table
                        then update pilot-rec;
                        else insert pilot-rec;
                end if;
        end repeat;
        end;
Process 2.3
        begin
                display all rank from rank table;
        end;
Process 2.4
        begin
        repeat until user exit the program
                select one from [all:wing:org];
                if selection = all then
                        display all pilot-rec from pilot order by pilot-id;
                        else if selection = wing then
                            display all pilot-rec of the wing from pilot order by pilot-id;
                            else if selection = org then
                                display all pilot-rec of the org from pilot order by pilot-id;
                end if;
        end repeat;
        end;
Process 2.5
        begin
        repeat until user exit the program
                select one from [all:wing];
                if selection = all
                        then compute pilot-number of cac;
                        * pilot-number = pilot-tot+hospt+vact+off+ready *
                        display pilot-number of cac;
                        repeat until data end of wing table;
                            compute pilot-number of the wing and their sqs;
                            display pilot-number of the wing and their sqs;
                        end repeat;
                        repeat until data end of other-dept table
                            compute pilot-number of the other-dept;
```

```
                display pilot-number of the other-dept;
            end repeat;
        end if;
        if selection = wing then
            reapeat until data end of wing table
                compute pilot-number of the wing;
                display pilot-number of the wing;
                repeat until data end of sq table;
                    compute pilot-number of the sq;
                    display pilot-number of the sq;
                end repeat;
            end repeat;
        end if;
    end repeat;
    end;
```

3.2.5 Partition of Aircraft. The Air Force has several types of aircraft and each *aircraft-type* is dictated a *grand-mission* (fight, carry, observe, or training). Each *aircraft-type* is located in one or more wing(s). Each *aircraft* has its own unique *ac-no* and *start-date* to run. Some organizations such as squadrons, and logistic command have *aircraft* while others do not. *Aircraft* can be transferred to another organization. Organizations which have *aircraft* must continuously count how many *aircraft* are on maintenance and how many *aircraft* are ready to take off to estimate the *ready-mission* status.

CAC is responsible for maintaining the information on *aircraft-type*, and each organization which has *aircraft* is responsible for maintaining the newest information on their own *aircraft*. Each organization can read the newest information of *aircraft-type* and *aircraft*. *CAC* reads the *aircraft-status* which shows each *aircraft-type*'s percentage of how many *aircraft* are ready to *take-off*. If the percentage is lower than a specific level, CAC does some act to increase the percentage. The following are process specifications of each functional primitive:

Process 3.1

```
    begin
    repeat until user exit the program
        accept ac-type;
        if the ac-type exists in ac-type table
            then display ac-type-rec;
        end if;
```

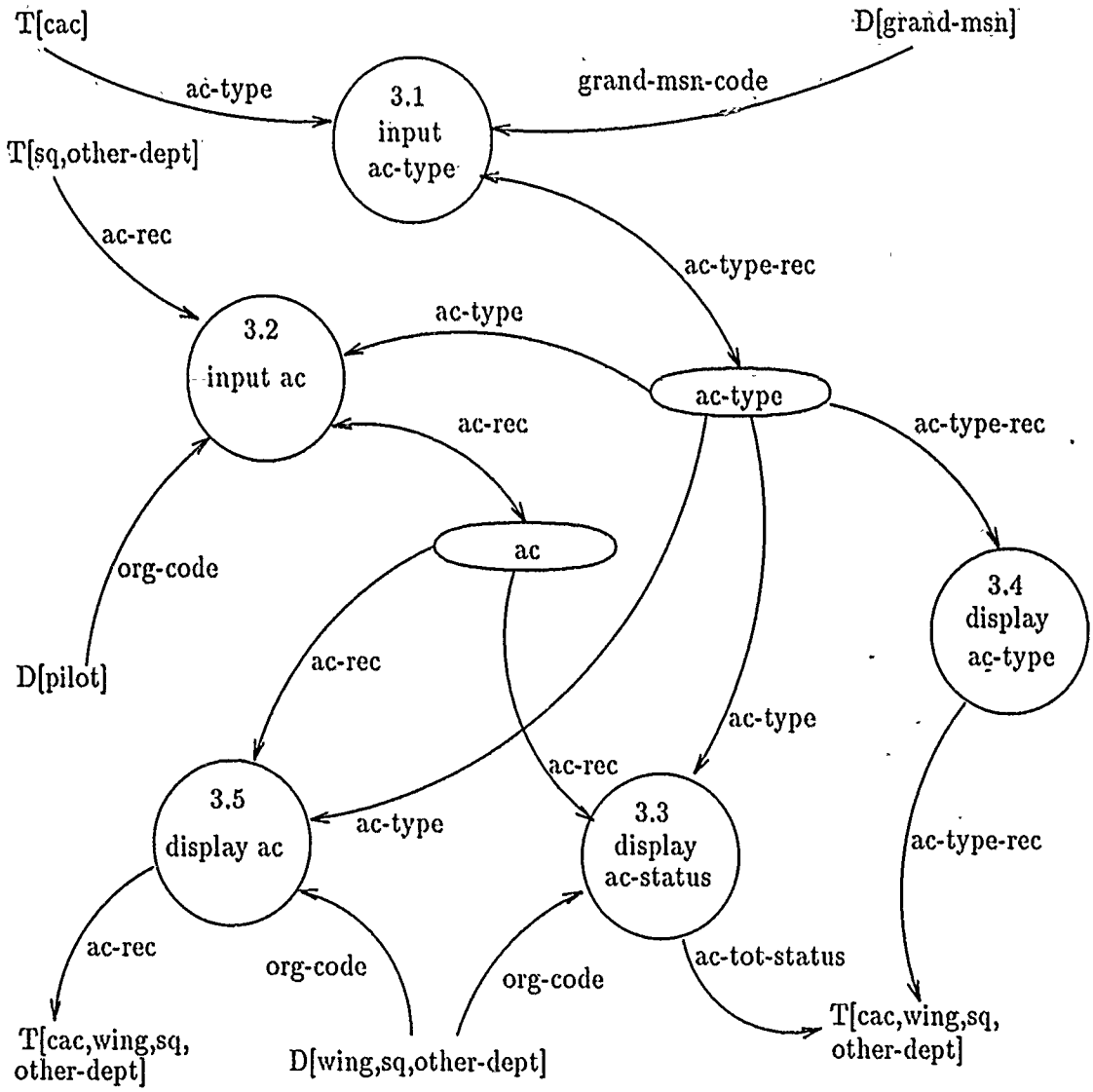


Figure 10. DFD of Aircraft

```
            accept ac-type-rec;
            check validation of grand-msn-code;
            if ac-type exists in ac-type table
                then update ac-type-rec;
                else insert ac-type-rec;
            end if;
        end repeat;
        end;
```

Process 3.2

```
        begin
        repeat until user exit the program
            accept ac-no;
            if ac-no exists in ac table
                then display ac-rec;
            end if;
            accept ac-rec;
            check validation of ac-type, org-code;
            if ac-no exists in ac table
                then update ac-rec;
                else insert ac-rec;
            end if;
        end repeat;
        end;
```

Process 3.3

```
        begin
        repeat until user exit the program
            select from [all-ac-type:one-ac-type];
            if selection = all-ac-type then
                repeatuntil data end of ac-type table
                    compute ac-tot-status of the ac-type;
                    * ac-tot-status=ac-rdy+ac-maint+ac-tot+ac-pct *
                    display ac-tot-status of the ac-type;
                end repeat;
            end if;
            if selection = one-ac-type then
                accept ac-type;
                repeat until data end of org
                    compute ac-tot-status of the org, of the ac-type;
                    display ac-tot-status of the org, of the ac-type;
                end repeat;
            end if;
        end repeat;
        end;
```

Process 3.4

```
        begin
```

```
            display all ac-type-rec from ac-type order by grand-msn-code, ac-type;
        end;
Process 3.5
        begin
        repeat until data end of ac-type table
            display all ac-rec of ac table;
        end repeat;
        end;
```

3.2.6 Partition of Mission. Flight *missions* of the Air Force can be grouped into four *grand-missions*: fight, carry, observe and train. Each *wing* and *aircraft-type* is charged with a *grand-mission*. These *grand-missions* are subdivided into several *missions* each of which is assigned to each *flight-order* and *flight-sortie*. An *exercise* is a set of military practices to prepare for an emergency situation. An *exercise* can be initiated by assigning some *flight-orders* which include a specific *flight-mission* to it. *Squadrons* transform the *flight-orders* into a *flight-plan* of a specific date.

CAC manages the contents of *grand-mission*, *mission*, and *exercise*. It may merge two or more items of each into one and may subdivide one item of each into two or more. Also, it may change the *grand-mission* of a certain *wing*. Each of the flying organizations refers to the *grand-mission*, *mission*, and *exercise* when they plan a flight. The following are process specifications of each functional primitive:

```
Process 4.1
        begin
        repeat until user exit the program
            accept grand-msn-code;
            if the grand-msn-code exists in grand-msn table
                then display grand-msn;
            end if;
            accept grand-msn;
            if grand-msn-code exists in grand-msn table;
                then update grand-msn;
                else insert grand-msn;
            end if;
        end repeat;
        end;
Process 4.2
```

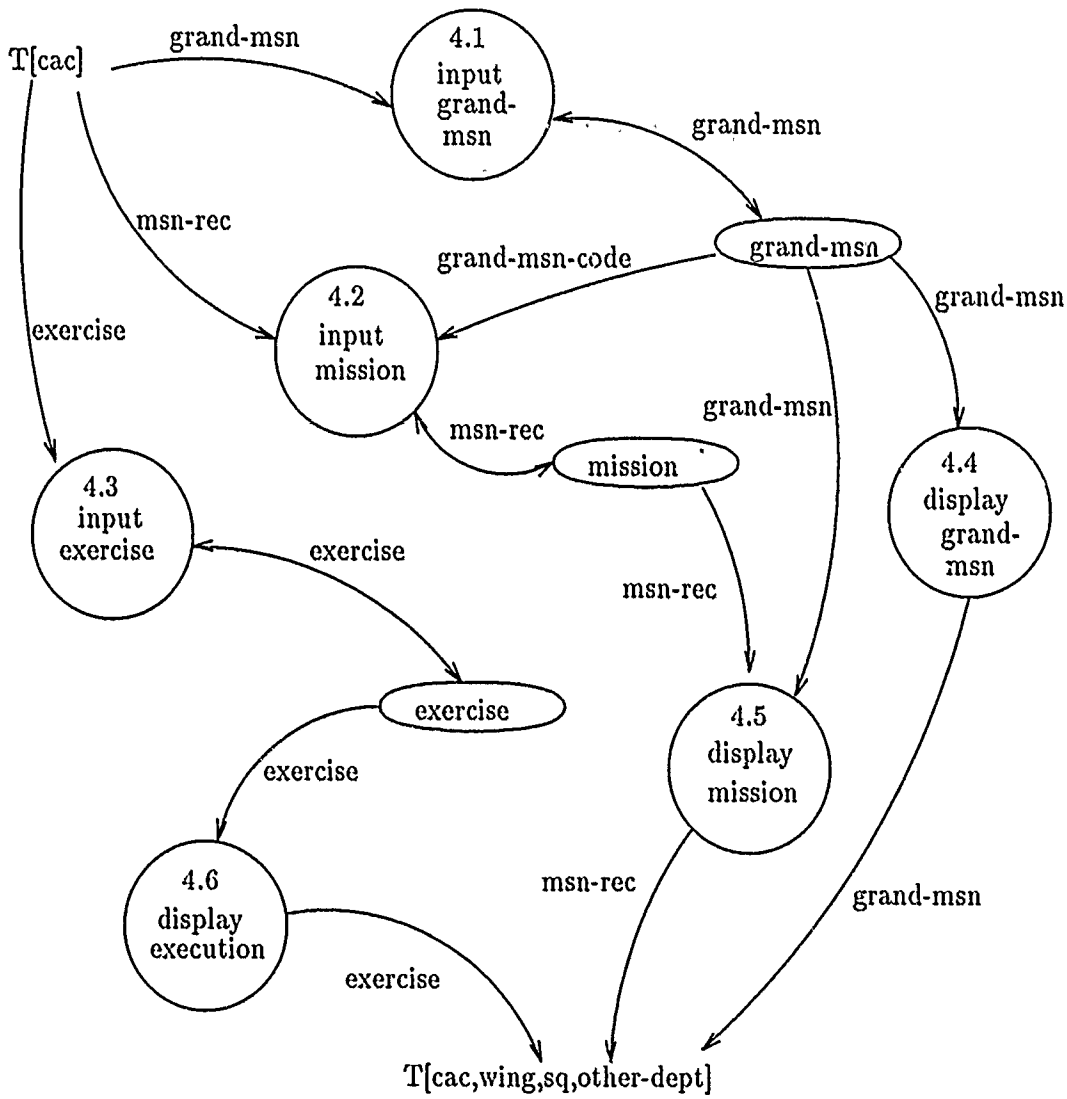


Figure 11. DFD of Mission

```
        begin
        repeat until user exit the program
                accept msn-code;
                if msn-code exists in mission table
                        then display msn-rec;
                end if;
                accept msn-rec;
                check validity of grand-msn-code;
                if msn-code exists in mission table
                        then update msn-rec;
                        else insert msn-rec;
                end if;
        end repeat;
        end;
```

Process 4.3

```
        begin
        repeat until user exit the program
                accept ex-code;
                if ex-code exists in exercise table
                        then display exercise;
                end if;
                accept exercise;
                if ex-code exists in exercise table
                        then update exercise;
                        else insert exercise;
                end if;
        end repeat;
        end;
```

Process 4.4

```
        begin
                display all grand-msn of grand-msn table order by grand-msn-code;
        end;
```

Process 4.5

```
        begin
                display all mission-rec of mission table order by msn-code;
        end;
```

Process 4.6

```
        begin
                display all exercise of exercise table order by ex-code;
        end;
```

3.2.7 Partition of Flight-order. There are two kinds of *flight-order: cac-order* and *wing-order. Cac-orders* are sent from *CAC* to each *wing* and *wing-orders* are sent from each *wing* to each of their *squadron.* The purpose of *flight-orders* are described in Section 1.2. A *flight-order* consists

of a *order-number*, receiving *organization-code*, *take-off* time, a *mission* code, an *exercise* code[1], an *aircraft-type*, *number-of-aircraft* required, and *description* of that *flight-order*.

CAC sends *cac-orders* to each *wing* continuously at any time. In the emergency situation, alarm system works before sending *flight-orders*. As soon as each *wing* receives a *cac-order*, they write *wing-order*(s) and send them to their *squadron*(s). Each *wing* may initiate *wing-orders* and send them to their *squadrons* at any time. Each *squadron* must perform *wing-orders* at the specified *take-off* time.

The FIS must support *CAC* to input and change *cac-orders* easily. *Cac-orders* of each *wing* must be displayed on the screen of each *wing* simultaneously as *CAC* inputs them. Like *cac-orders*, the FIS must support each *wing* inserting and changing *wing-orders* easily. *Wing-orders* of each *squadron* must be displayed on the screen of each *squadron* simultaneously as *wing* inputs them. Since *flight-orders* are sent at unexpected times, they must be displayed automatically at the screen of destination. Each *wing* deletes a received *cac-order* after reordering it to their *squadron*(s) through *wing-order*(s). Similarly, each *squadron* deletes a received *wing-order* after translating it to *flight-plan*(s) which is to be executed by *pilots*. The following are process specifications of each functional primitive:

Process 5.1

```
begin
repeat until user exit the program
        accept cac-order-no;
        if the cac-order-no exists in cac-order table
                then display cac-order-rec;
        end if;
        accept cac-order-rec;
        check validity of ac-type, ex-code, msn-code, and wing-code;
        if cac-order-no exists in cac-order table
                then update cac-order-rec;
                else insert cac-order-rec;
        end if;
        update change-time of data-change table to current time;
end repeat;
end;
```

[1] An *exercise* code is required only if the *flight-order* is for an *exercise*

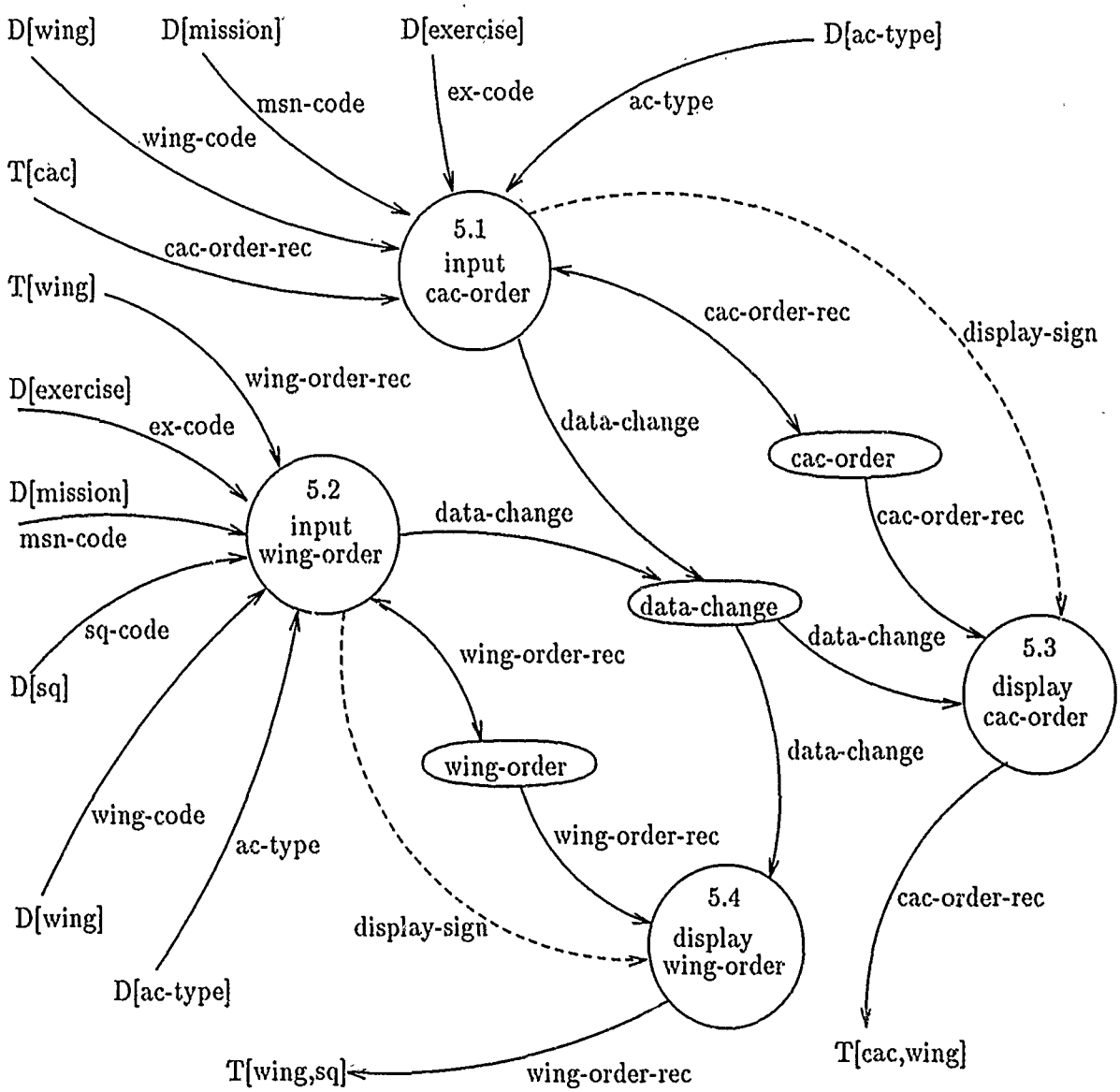


Figure 12. DFD of Flight-order

Process 5.2

```
begin
repeat until user exit the program
        accept wing-code + wing-order-no;
        if wing-code + wing-order-no exists in wing-order table
                then display wing-order-rec;
        end if;
        accept wing-order-rec;
        check validity of ac-type, ex-code, msn-code, sq-code, and wing-code;
        if wing-code + wing-order-no exists in wing-order table
                then update wing-order-rec;
                else insert wing-order-rec;
        end if;
        update change-time of data-change table to current time;
end repeat;
end;
```

Process 5.3

```
begin
time := 0;
accept wing-code;
repeat until user exit the program
        read change-time of the wing-code from data-change table;
        if change − time > time
                then display cac-order of the wing-code from cac-order table;
        end if;
        time := change-time;
end repeat;
end;
```

Process 5.4

```
begin
time := 0;
accept cac-code;
repeat until user exit the program;
        read change-time of the cac-code from data-change table;
        if change − time > time
                then display wing-order of the cac-code from wing-order table;
        end if;
        time := change-time;
end repeat;
end;
```

3.2.8 Partition of Flight-sortie. Each flying organization writes *flight-plans* to perform *wing-orders* or to perform its *missions*. A *flight-plan* consists of a *set-sortie* and one or more sorties. The contents of *set-sortie* and *sortie* are shown in DD of Appendix A. A *flight-plan* can be added,

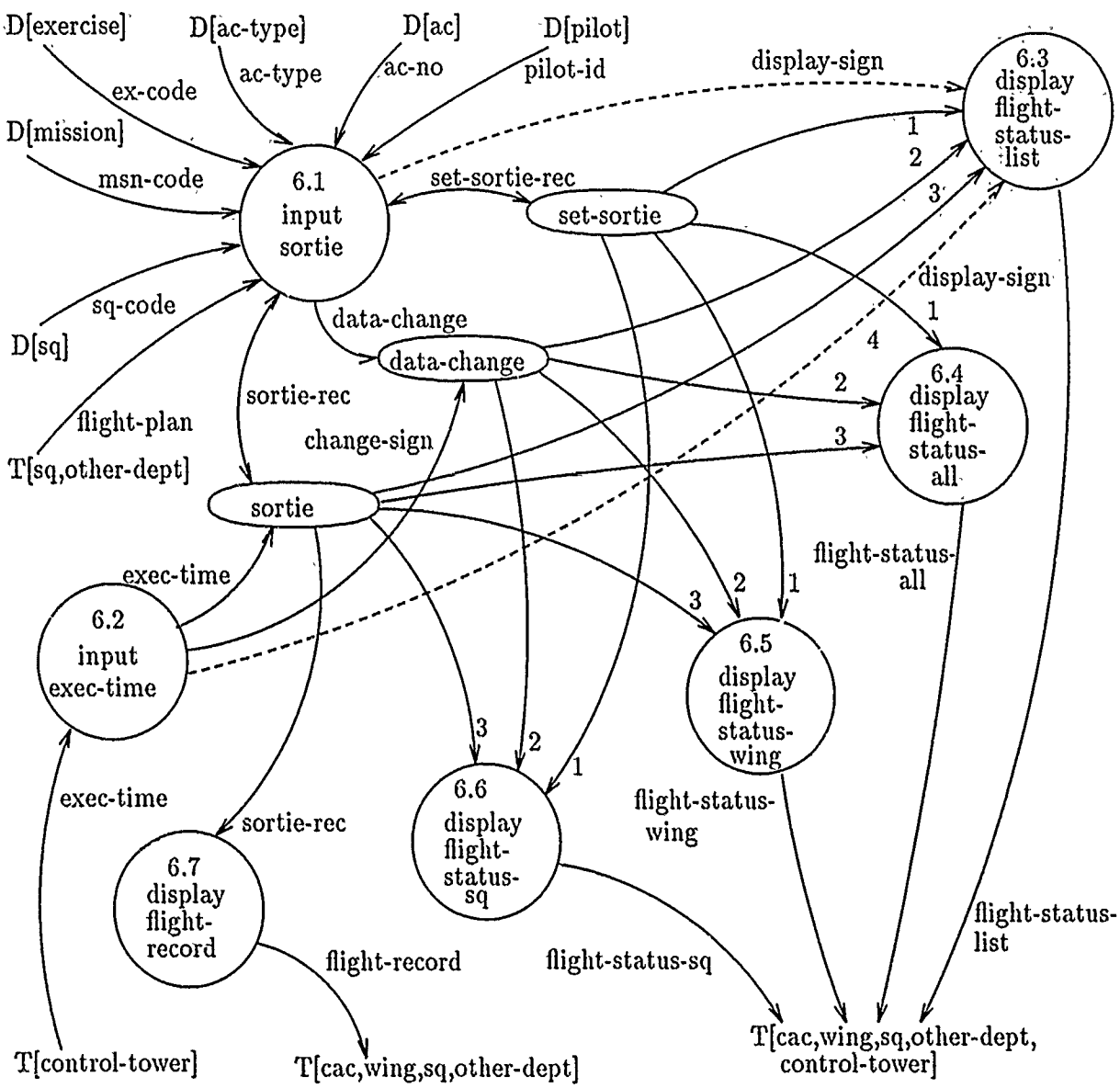
delayed, or canceled continuously because of the weather and *wing-order*. The *flight-plan* is very important for the safety of flight. For example, planned *take-off* time of an organization's *flight-plan* must be avoided by other organizations's *flight-plan* when they share a runway to avoid aircraft collision. Therefore, all information related to *flight-plan* must be shared by all organizations of the FIS.

Each air base has a *control-tower* and one or more *wings*. A *control-tower* controls and inputs the *take-off* and *landing-time* of all *aircraft* which use its runway or airfield. Each *control-tower* can anticipate the *take-off* and *landing* of *aircraft* not only through communication with *aircraft* but also through the computer screen displaying *flight-plan* of each organization.

All the information related to a *flight-sortie* is required to be processed simultaneously as the input from each organizations, and all the information related on *flight-sorties* required to be shared by all organizations of the FIS. Thus, all organizations of the FIS can read detailed, reliable, and current information on *flight-sorties* on the computer screen. For example, *flight-status* of an organization, of a *wing*, or of the Air Force can be known easily. Also, flight record of a pilot or of a group can be known easily. The following are process specifications of each functional primitive:

Process 6.1

```
begin
repeat until user exit the program
    accept plan-take-off + sq-code + set-no;
    if the plan-take-off+sq-code+set-no exists in set-sortie table
        then display set-sortie-rec;
    end if;
    accept set-sortie-rec;
    check validity of ac-type, ex-code, msn-code, and org-code;
    if plan-take-off+sq-code+set-no exists in set-sortie table
        then update set-sortie table;
        else insert set-sortie table;
    end if;
    update change-time of data-change table to current time;
    repeat until user exit the routine;
        accept plan-take-off+sq-code+set-no+position-no;
        if the plan-take-off+sq-code+set-no+position-no exists in sortie table
            then display sortie-rec;
        end if;
        accept sortie-rec;
```



*COMMENTS

-1,2, and 3 denotes set-sortie-rec, data-change, and sortie-rec respectively.

Figure 13. DFD of Flight-sortie

```
                    check validity of ac-no and pilot-id;
                    if plan-take-off+sq-code+set-no+position-no exists in sortie table
                        then update sortie table;
                        else insert sortie table;
                    end if;
                    update change-time of data-change table to current time;
                end repeat;
            end repeat;
            end;
Process 6.2
            begin
            repeat until user exit the program
                update take-off or update landing of sortie table;
                update change-time of data-change table;
            end repeat;
            end;
Process 6.3
            begin
            time := 0
            repeat until user exit the program;
                read change-time of data-change table;
                if change − time > time then
                    repeat until data end of wing table
                        display wing-code of wing table;
                        repeat until data end of sq table
                            display sq-code of sq table;
                            compute flight-status-list of the sq;
                            display flight-status-list of the sq;
                        end repeat;
                    end repeat;
                end if;
                time := change-time;
            end repeat;
            end;
Process 6.4
            begin
                display all set-sorties from set-sortie table;
                display all sorties from sortie table;
            end;
Process 6.5
            begin
                accept wing-code;
                display all set-sorties of the wing-code from set-sortie table;
                display all sorties of the wing-code from sortie table;
            end;
```

Process 6.6

begin
accept *sq-code*;
display all *set-sorties* of the *sq-code* from *set-sortie* table;
display all *sorties* of the *sq-code* from *sortie* table;
end;

Process 6.7

begin
accept conditions (*from-date, to-date, org-code*, and/or *pilot-id*);
display all *sorties* which satisfy the conditions from *sortie* table;
end;

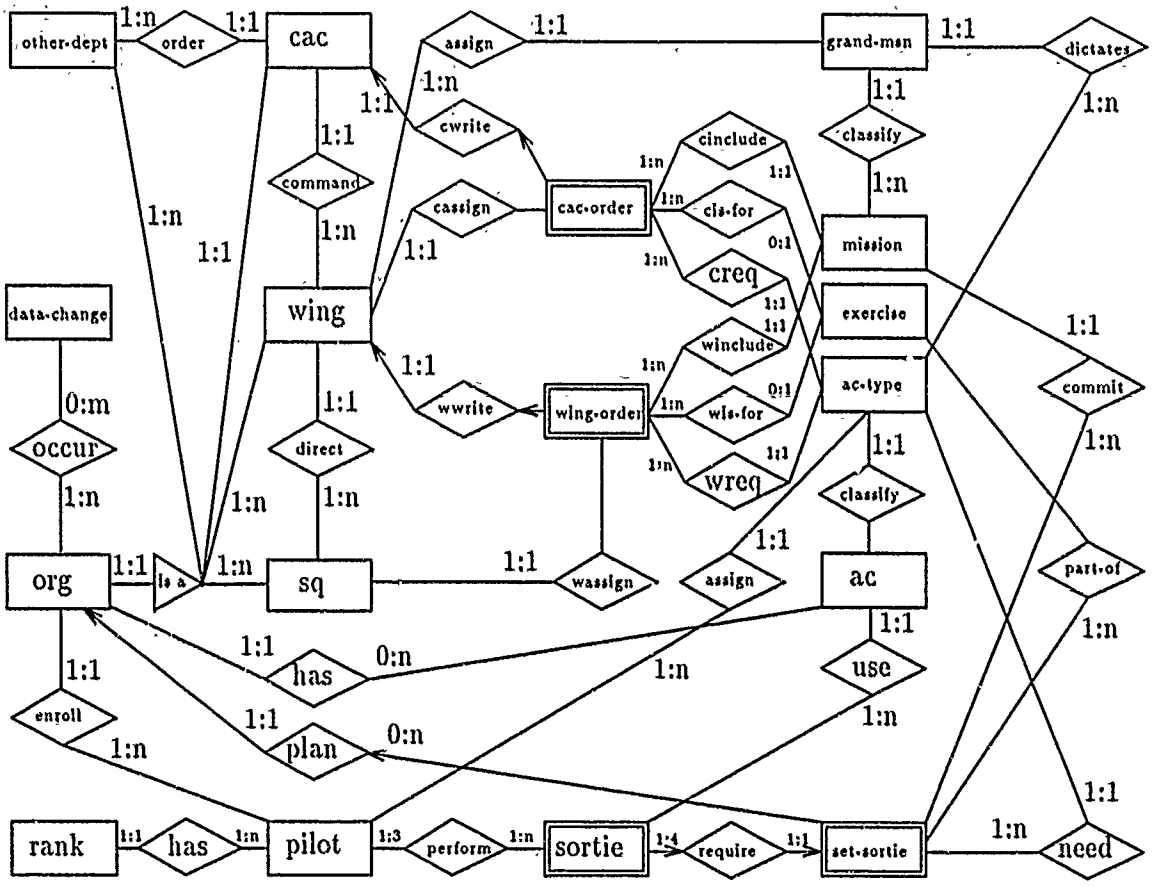
3.3 Database Modelling

This section presents the process of database modelling of the FIS using DFDs and DD which were generated from the last section. ERD is a tool to be used in this section. All entities, relationships, and their attributes of the FIS will be described on the ERDs. Comments which are difficult to describe by entities and/or relationships will be described by short English on the bottom part of the ERDs.

Figure 14 shows an overview of the ERD for the FIS. The ERD of the FIS can be grouped into six major parts: *organization, pilot, aircraft, mission, flight order*, and *flight-sortie* depend on the characteristic of each entities. The scope of each part is similar to the scope of corresponding process of Figure 7 which was decomposed from the context diagram. Each of the six parts will be discussed in detail in this section. The entity sets and weak entity sets of each part are shown in Table 3.

No	Part	Entity Sets & Weak Entity Sets
1	organization	cac, wing, sq, org, other-dept, data-change
2	pilot	pilot, rank
3	aircraft	ac, ac-type
4	mission	grand-msn, mission
5	flight-order	*cac-order*[1] *wing-order*, exercise
6	flight-sortie	*set-sortie, sortie*

Table 3. Entity/Weak Entity Sets of the FIS

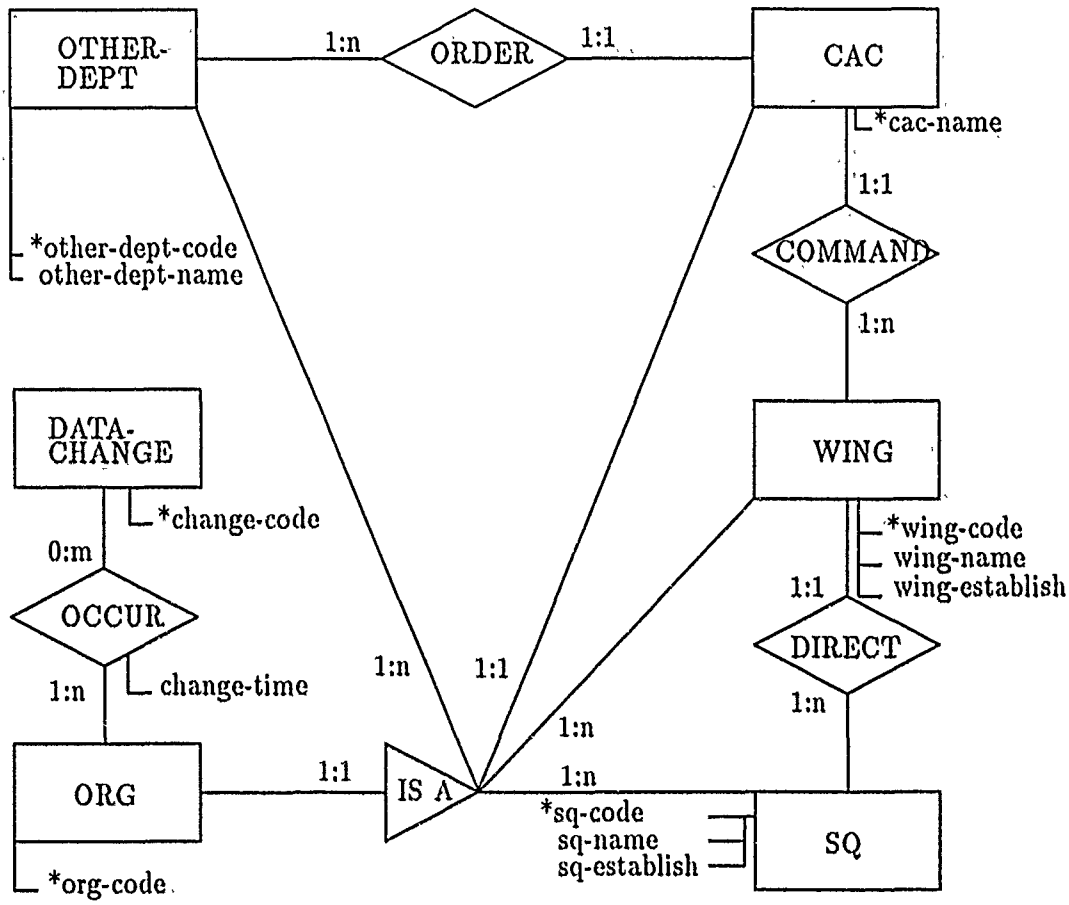


* COMMENTS

-Data-changes occur at wing and organizations which perform flight.
-Pilots promote to another rank ocassionally.
-Pilots can be assigned to another AC-TYPE.
-The role of each pilot in a sortie is different.

Figure 14. Overview ERD of the FIS

3.3.1 Modelling of Organization. Figure 15 shows the ERD of organization. The ERD of organization consists of six entity sets, four relationship sets, and their attributes. Entity sets and their attributes of the organization are shown in Table 4.



* COMMENTS

-Data-changes occur at wing and organizations which perform flight.

Figure 15. ERD of Organizations

Org is an entity set produced by generalizing four entity sets *cac, wing, sq*, and *other-dept.*. Entity set *org* makes it possible to validate an organization code easily. *Data-change* is an entity set

Entity/Weak Entity Set	Occurrence[1]	Attributes
cac	1	*cac-name* [2]
wing	50	*wing-code*, wing-name, wing-establish
sq	200	*sq-code*, sq-name, sq-establish
other-dept	50	*other-dept-code*, other-dept-name
org	500	*org-code*
data-change	4	*change-code*

Table 4. Entity/Weak Entity Sets & Attributes of Organization

required to read the newest change time of certain data of certain an organization. Relationships between entities and further descriptions which are not shown on the ERD are described below.

1. *Cac* commands all *wings*.
2. A *wing* directs their *squadrons*.
3. *Cac* orders each pilot of *other-dept* to move to certain *org* in emergency case.
4. *Data-change*[3] occurs at *wing* and *orgs* which perform flights.

3.3.2 Modelling of Pilot. Figure 16 shows the ERD of pilot. The ERD of pilot consists of two entity sets and three relationship sets and their attributes. Entity sets and their attributes are described in Table 5.

Entity/Weak Entity Set	Occurrence	Attributes
pilot	10,000	*pilot-id*, name, class, blood-type, pilot-date, job, pilot-status
rank	50	*rank-code*, rank-name

Table 5. Entity/Weak Entity Sets & Attributes of Pilot

Relationships between entities and further descriptions which are not shown on the ERD are described below.

[1] Occurrence means the maximum number of the occurrence of each entity/weak entity.
[2] Primary key attributes are italicized.
[3] *Data-change* includes receiving *flight-order*, changing *flight-plan*, and/or executing *flight-sortie*.

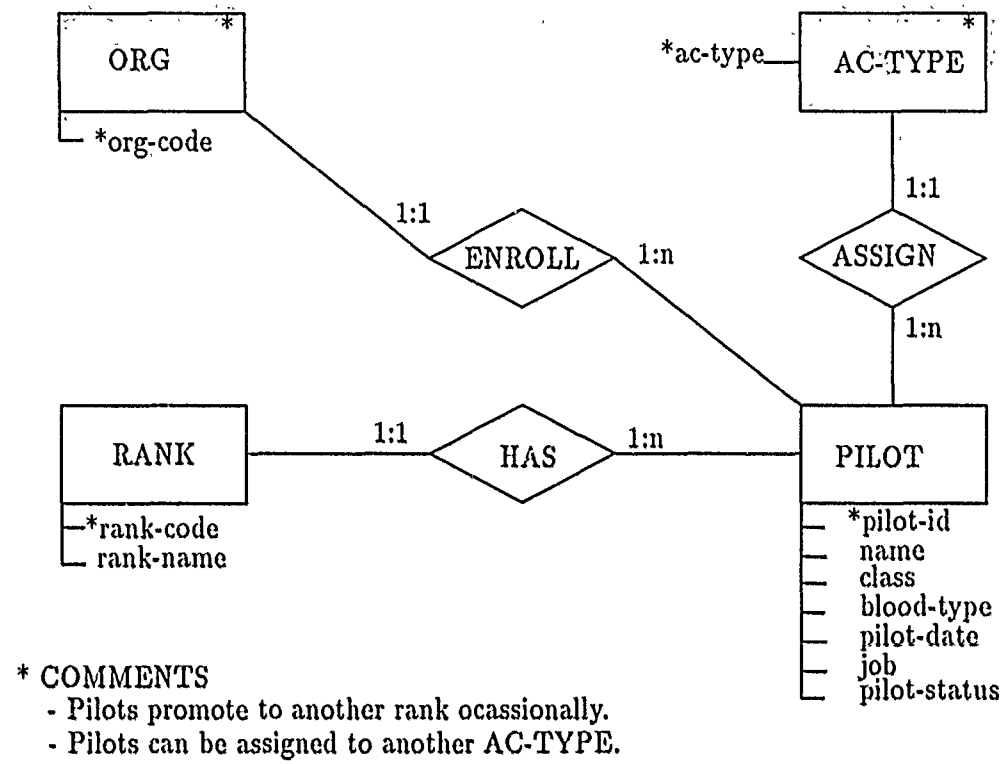


Figure 16. ERD of Pilot

1. A *pilot* has a *rank* and he is promoted to another *rank* occasionally.
2. A *pilot* is enrolled to an *org* and he may move to another *org*.
3. A *pilot* is assigned to control a specified *ac-type*.

3.3.3 Modelling of Aircraft. Figure 17 shows the ERD of aircraft which consists of two entity sets and two relationship sets. Entity sets and their attributes are shown in Table 6.

Entity/Weak Entity Set	Occurrence	Attributes
ac-type	50	*ac-type*
ac	5,000	*ac-no*, ac-status, start-date

Table 6. Entity/Weak Entity Sets & Attributes of Aircraft

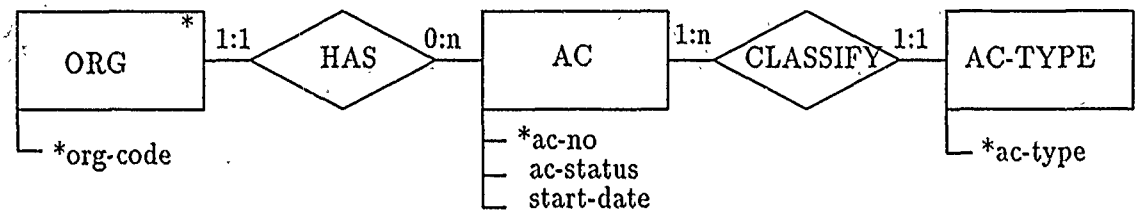


Figure 17. ERD of Aircraft

Relationships between entities and further descriptions which are not shown on the ERD are described below.

1. *Acs* are classified into many *ac-types*.
2. Some *orgs* have *acs*.

3.3.4 Modelling of Mission. Figure 18 shows the ERD of mission which consists of two entity sets and three relationship sets. Entity sets and their attributes are shown in Table 7.

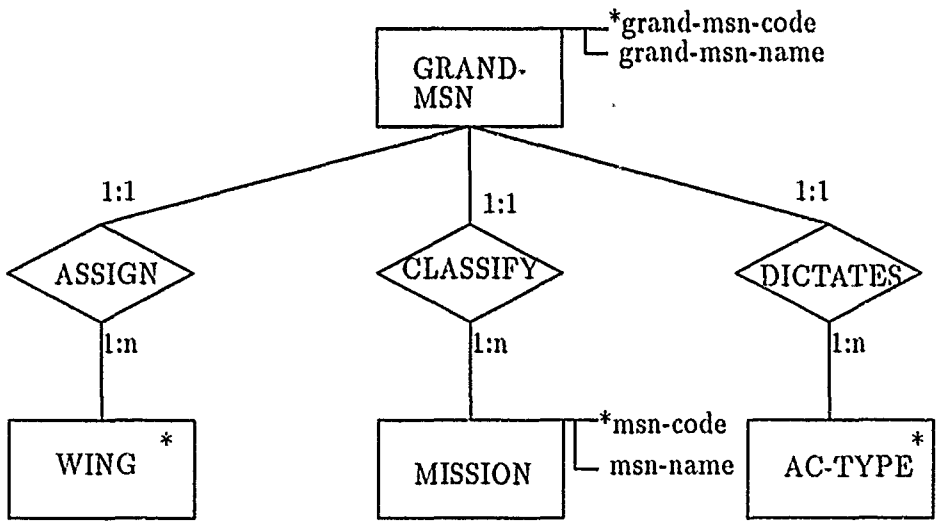


Figure 18. ERD of Mission

Relationships between entities and further descriptions which are not shown on the ERD are described below.

1. A *wing* is assigned to a *grand-msn*.

Entity/Weak Entity Set	Occurrence	Attributes
grand-msn	10	*grand-msn-code*, grand-msn-name
mission	1,000	*msn-code*, msn-name

Table 7. Entity/Weak Entity Sets & Attributes of Mission

2. An *ac-type* is dictated a *grand-msn.*

3. A *grand-msn* is classified into many *missions.*

3.3.5 Modelling of Flight-order. Figure 19 shows the ERD of flight-order which consists of one entity set, two weak entity sets, and ten relationship sets. The existence of a weak entity *cac-order* depends on the existence of a strong entity *cac.* Also, the existence of another weak entity *wing-order* depends on the existence of a strong entity *wing.* Entities and their attributes are shown in Table 8.

Entity/Weak Entity Set	Occurrence	Attributes
exercise	100	*ex-code*, ex-name
cac-order[1]	1,000	cac-order-no, number-ac, take-off, cac-descript
wing-order	1,000	wing-order-no, take-off, wing-descript

Table 8. Entity/Weak Entity Sets & Attributes of Flight-Order

Relationships between entities and further descriptions which are not shown on the ERD are described below.

1. *Cac* writes *cac-orders.*

2. A *cac-order* includes a *mission.*

3. A *cac-order* require an *ac-type.*

4. A *cac-order* may be for an *exercise.*

5. A *cac-order* is assigned to a *wing.*

[1]Weak entity sets are italicized.

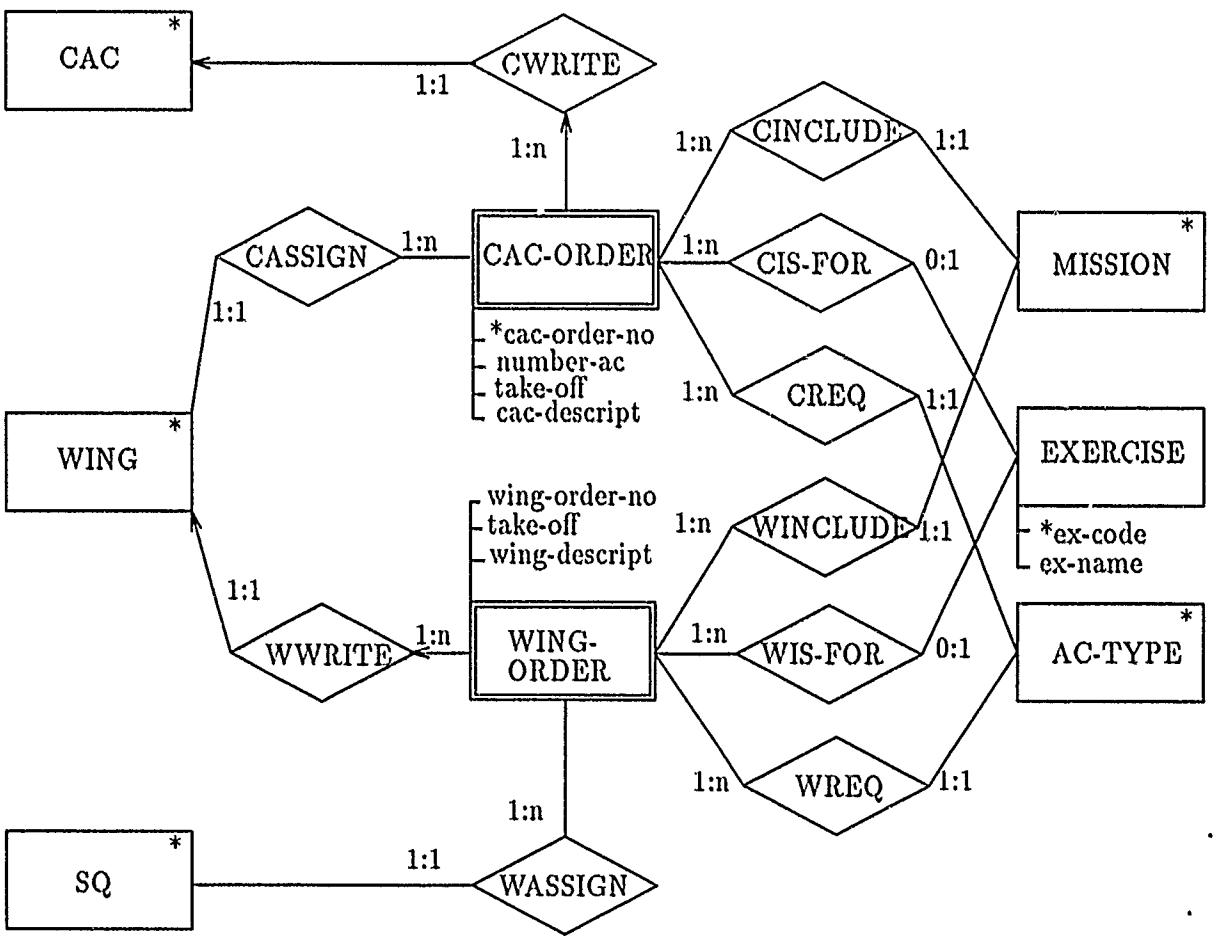


Figure 19. ERD of Flight-order

6. A *wing* writes *wing-orders*.

7. A *wing-order* includes a *mission*.

8. A *wing-order* require an *ac-type*.

9. A *wing-order* may be for an *exercise*.

10. A *wing-order* is assigned to a *sq*.

3.3.6 Modelling of Flight-sortie. Figure 20 shows the ERD of flight-sortie which consists of two weak entity sets and seven relationship sets. The existence of a weak entity *set-sortie* is dependent on the strong entity *org*. The existence of another weak entity *sortie* is dependent on the weak entity *set-sortie* and the strong-entity *org*. Entity sets and their attributes are shown in Table 9.

Entity/Weak Entity Set	Occurrence	Attributes
set-sortie	500,000	set-no, plan-take-off
sortie	1,500,000	position-no, take-off, landing

Table 9. Entity/Weak Entity Sets & Attributes of Flight-Sortie

Relationships between entities and further descriptions which are not shown on the ERD are described below.

1. Some *org* plan *set-sortie*.

2. A *set-sortie* needs an *ac-type*.

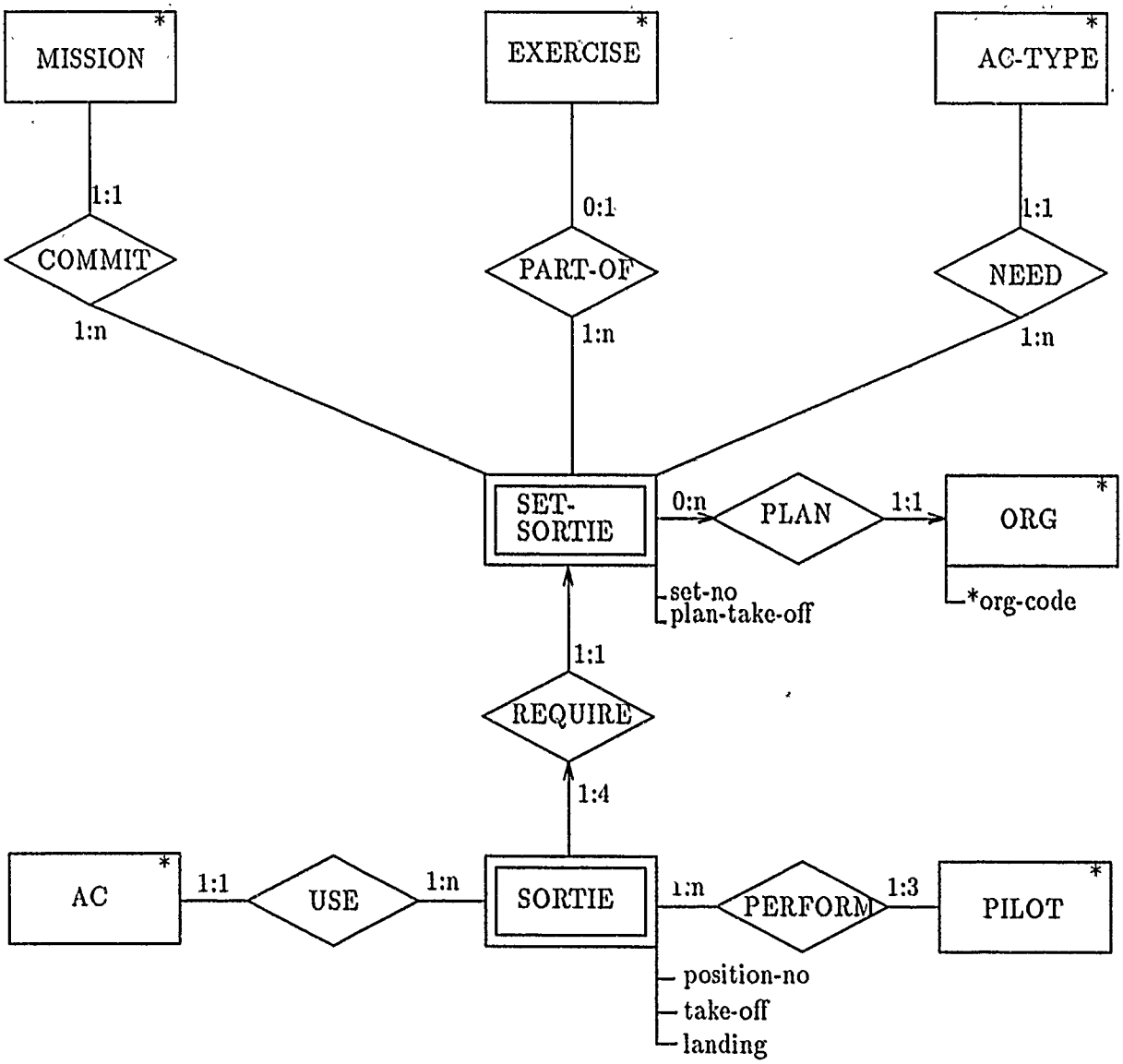
3. A *set-sortie* may or may not be a part of an *exercise*.

4. A *set-sortie* commits a *mission*.

5. A *set-sortie* require one, two , three, or four *sorties*.

6. A *sortie* is performed by one, two, or three *pilots*.

7. A *sortie* uses an *ac*.



* COMMENTS

- The role of each pilot in a sortie is different.

Figure 20. ERD of Flight-sortie

3.4 Summary

This chapter presented an analysis of the FIS using the structured analysis technique. User required data was identified by using the DFD, DD, and process specification tools. Database modelling activity is followed to generate the structure of a database from the perception of the real world of the FIS. ERD was used as a tool in this step.

IV. Design of the FIS

4.1 Introduction

This chapter describes the design of the FIS using the structured design technique. The database will be designed from the DD and ERD which were produced in Chapter 3. After that, structure charts will be derived from the DFD which were also produced in Chapter 3. The module design which describes the implementation method of each module will be described to aid the programming activity.

4.2 Database Design

This section shows how the relational database tables are generated from the DD and ERD. Normalization techniques will be used to generate the tables to a high degree of integrity and maintainability. This thesis sets two goals for the database design. One of the goals is achieving BCNF in the normalization. If it is not possible, however, 3NF is acceptable. Another goal of the database design in this thesis is to generate an appropriate number of database tables. Too large a number of database tables increases the complexity of the database system and requires redundant efforts for development. Too small a number of database tables, on the other hand, increases the size of a table and the computational cost.

To achieve the above goals, two steps will be performed. The first step is required to achieve the normalization goal. All entity sets and relationship sets of the ERD will be transformed into database tables. Then, these database tables will be transformed into the form which satisfy BCNF or 3NF. The second step is to decrease the number of tables which are generated in the first step. Some tables generated in the first step can be rejected or merged into a neighboring table to simplify the database system. For example, a table which has only one tuple can be rejected and a table which was generated by borrowing primary key attributes of each neighboring table can be merged into one of the neighboring tables.

4.2.1 Database Design of Organization. Each of the entity sets and relationship sets of organization shown in Figure 15 are transformed into tables as shown in Table 10. All of the tables satisfy BCNF.

Table	Attributes & Primary key
cac	*cac-name*[1]
command	*wing-code*, cac-name
data-change	*change-code*
direct	*sq-code*, wing-code
occur	*org-code, change-code*, change-time
order	*other-dept-code*, cac-name
org	*org-code*
other-dept	*other-dept-code*, other-dept-name
sq	*sq-code*, sq-name, sq-establish
wing	*wing-code*, wing-name, wing-establish

Table 10. Incomplete Database Tables of Organization

Tables *cac, command*, and *order* need to be rejected since table *cac* has only one tuple. Table *direct* can be merged into table *sq* since each of those tables has the same primary key *sq-code*. Also, table *data-change* can be merged into table *occur* since table *data-change* is a subset of table *occur*. Since table *org* is a generalization of tables *cac, wing, sq, and other-dept*, and has other general attributes other than the *org-code*, there is no need for it to exist as a table. Table *assign* of Section 4.2.4 is merged into table *wing* since the primary key of each is the same. The complete database tables of organization are shown in Table 11.

Table	Attributes & Primary key
occur	*org-code, change-code*, change-time
other-dept	*other-dept-code*, other-dept-name
sq	*sq-code*, sq-name, sq-establish, wing-code
wing	*wing-code*, wing-name, wing-establish, grand-msn-code

Table 11. Complete Database Tables of Organization

4.2.2 Database Design of Pilot. Each of the entity sets and relationship sets of pilot, shown in Figure 16, are transformed into a table as shown in Table 12. All of the tables satisfy BCNF. Tables *enroll, has*, and *assign* are merged into a table *pilot* since the primary key of each is the

[1]Italicized attribute(s) denote primary key of the database table

Table	Attributes & Primary key
assign	*pilot-id*, ac-type
enroll	*pilot-id*, org-code
has	*pilot-id*, rank-code
pilot	*pilot-id*, name, class, blood-type, pilot-date, job, pilot-status
rank	*rank-code*, rank-name

Table 12. Incomplete Database Tables of Pilot

same. Table 13 shows the complete database tables generated from pilot.

Table	Attributes & Primary key
pilot	*pilot-id*, name, class, blood-type, pilot-date, job, pilot-status, rank-code, org-code, ac-type
rank	*rank-code*, rank-name

Table 13. Complete Database Tables of Pilot

4.2.3 Database Design of Aircraft. Each of the entity sets and relationship sets of aircraft, shown in Figure 17, are transformed into a table as shown in Table 14. All of the tables satisfy BCNF. Tables *classify* and *has* are merged into a table *ac* since the primary key of each is the same.

Table	Attributes & Primary key
ac	*ac-no*, ac-status, start-date
ac-type	*ac-type*
classify	*ac-no*, ac-type
has	*ac-no*, org-code

Table 14. Incomplete Database Tables of Aircraft

Table *dictate* of Section 4.2.4 is merged into table *ac-type* since the primary keys of each are the same. Table 15 shows the complete database tables generated from *aircraft*.

Table	Attributes & Primary key
ac	*ac-no*, ac-status, start-date, ac-type, org-code
ac-type	*ac-type*, grand-msn-code

Table 15. Complete Database Tables of Aircraft

4.2.4 Database Design of Mission. Each of the entity sets and relationship sets of mission, shown in Figure 18, are transformed into a database table as shown in Table 16. All of the tables satisfy BCNF.

Table	Attributes & Primary key
assign	*wing-code*, grand-msn-code
classify	*msn-code*, grand-msn-code
dictates	*ac-type*, grand-msn-code
grand-msn	*grand-msn-code*, grand-msn-name
mission	*msn-code*, msn-name

Table 16. Incomplete Database Tables of Mission

Tables *assign, classify,* and *dictates* are merged into table *wing, mission,* and *ac-type* respectively, since the primary key of each is the same. Table 17 shows the complete database tables generated from *mission.*

Table	Attributes & Primary key
grand-msn	*grand-msn-code*, grand-msn-name
mission	*msn-code*, msn-name

Table 17. Complete Database Tables of Mission

4.2.5 Database Design of Flight-order. Each of the entity sets and relationship sets of flight-order, shown in Figure 19, are transformed into a table as shown in Table 18. Table *wing-order* imported a key *wing-code* from the entity set *wing* which is the strong entity of weak entity set *wing-order.* Table *cac-order,* however, did not import a key since the entity set *cac* has only one entity value. All of the tables satisfy BCNF. Table *cwrite* is regarded to be a database table since

Table	Attributes & Primary key
cac-order	*cac-order-no*, number-ac, take-off, cac-descript
cassign	*cac-order-no*, wing-code
cinclude	*cac-order-no*, msn-code
cis-for	*cac-order-no*, ex-code
creq	*cac-order-no*, ac-type
cwrite	*cac-order-no*, cac-name
exercise	*ex-code*, ex-name
wassign	*wing-code, wing-order-no*, sq-code
winclude	*wing-code, wing-order-no*, msn-code
wing-order	*wing-code, wing-order-no*, take-off, wing-descript
wis-for	*wing-code, wing-order-no*, ex-code
wreq	*wing-code, wing-order-no*, ac-type
wwrite	*wing-code, wing-order-no*

Table 18. Incomplete Database Tables of Flight-order

the entity set *cac* has only one entity value. Tables *cassign, cinclude, cis-for,* and *creq* are merged into table *cac-order* since the primary key of each is the same. Similarly, tables *wassign, winclude, wis-for,* and *wreq* are merged into table *wing-order* for the same reason. Table 19 shows the complete database tables generated from *flight-order.*

Table	Attributes & Primary key
cac-order	*cac-order-no*, number-ac, take-off, cac-descript, wing-code, msn-code, ex-code, ac-type
exercise	*ex-code*, ex-name
wing-order	*wing-code, wing-order-no*, take-off, wing-descript, sq-code, msn-code, ex-code, ac-type

Table 19. Complete Database Tables of Flight-order

4.2.6 Database Design of Flight-sortie. Each of the entity sets and relationship sets of *flight-sortie*, shown in Figure 19, are transformed into a table as shown in Table 20. Table *set-sortie* imports a key from the entity set *org* which is a strong entity set of weak entity set *set-sortie.* Also, table *sortie* imports a key from entities *set-sortie* and *org* which are strong entity sets of weak entity set *sortie.* All of the tables satisfy BCNF. Tables *commit, part-of, need,* and *plan* are merged into

Table	Attributes & Primary key
commit	*org-code, set-no, plan-take-off*, msn-code
need	*org-code, set-no, plan-take-off*, ac-type
part-of	*org-code, set-no, plan-take-off*, ex-code
perform	*org-code, set-no, plan-take-off, position-no*, take-off, landing, front-pilot, rear-pilot, stand-by-pilot
plan	*org-code, set-no, plan-take-off*
require	*org-code, set-no, plan-take-off, position-no*
set-sortie	*org-code, set-no, plan-take-off*
sortie	*org-code, set-no, plan-take-off, position-no*, take-off, landing
use	*org-code, set-no, plan-take-off, position-no*, take-off, landing, ac-no

Table 20. Incomplete Database Tables of Flight-sortie

table *set-sortie* since the primary key of each is the same. Also, tables *require, perform,* and *use* are merged into table *sortie* since the primary key of each is the same. Table 21 shows the complete tables generated from *flight-sortie.*

Table	Attributes & Primary key
set-sortie	*org-code, set-no, plan-take-off*, msn-code, ex-code, ac-type
sortie	*org-code, set-no, plan-take-off, position-no*, take-off, landing, ac-no, front-pilot, rear-pilot, stand-by-pilot

Table 21. Complete Database Tables of Flight-sortie

4.2.7 Complete Database Tables. Table 22 shows the complete database tables generated from Section 4.2.1 through 4.2.6. It consists of 15 tables and their attributes. Appendix B shows the database creation program which was written by SQL language.

4.3 Structure Chart

This section describes how the structure chart of the FIS from the DFD is generated. Top-down design technique is used. The overview DFD of the FIS shown in Figure 7 shows that the FIS consists of six sub-systems. Figure 7 is transformed into a high-level structure chart and the

No	Table	Attributes & Primary key
1	ac	*ac-no*, ac-status, start-date, ac-type, org-code
2	ac-type	*ac-type*, grand-msn-code
3	cac-order	*cac-order-no*, number-ac, take-off, cac-descript, wing-code, msn-code, ex-code, ac-type
4	exercise	*ex-code*, ex-name
5	grand-msn	*grand-msn-code*, grand-msn-name
6	mission	*msn-code*, msn-name
7	occur	*org-code, change-code*, change-time
8	other-dept	*other-dept-code*, other-dept-name
9	pilot	*pilot-id*, name, class, blood-type, pilot-date, job, pilot-status, rank-code, org-code, ac-type
10	rank	*rank-code*, rank-name
11	set-sortie	*org-code, set-no, plan-take-off*, msn-code, ex-code, ac-type
12	sortie	*org-code, set-no, plan-take-off, position-no*, take-off, landing, ac-no, front-pilot, rear-pilot, stand-by-pilot
13	sq	*sq-code*, sq-name, sq-establish, wing-code
14	wing	*wing-code*, wing-name, wing-establish, grand-msn-code
15	wing-order	*wing-code, wing-order-no*, take-off, wing-descript, sq-code, msn-code, ex-code, ac-type

Table 22. Complete Database Tables of the FIS

sub-DFDs of Figure 7 are transformed into low-level structure charts. The functions of each module of structure chart are described in detail.

4.3.1 High-Level Design. The overview-DFD of the FIS shown in Figure 7 is transformed into a high-level structure chart of the FIS. The high-level structure chart is shown in Figure 21. Module 0 calls each of the sub-modules and each of the sub-modules calls its sub-modules also. The high-level modules, module 0 through module 6, need not exchange any data between is sub modules. Those modules will be developed using SQL*Menu. The functions of each high-level module described below are similar since the functions of each are just to call its sub-modules.

Modules 0, 1, 2, 3, 4, 5, and 6
begin
repeat until user exit the program
    display menu on the screen;

```
    select a menu;
    call selected menu;
end repeat;
end;
```

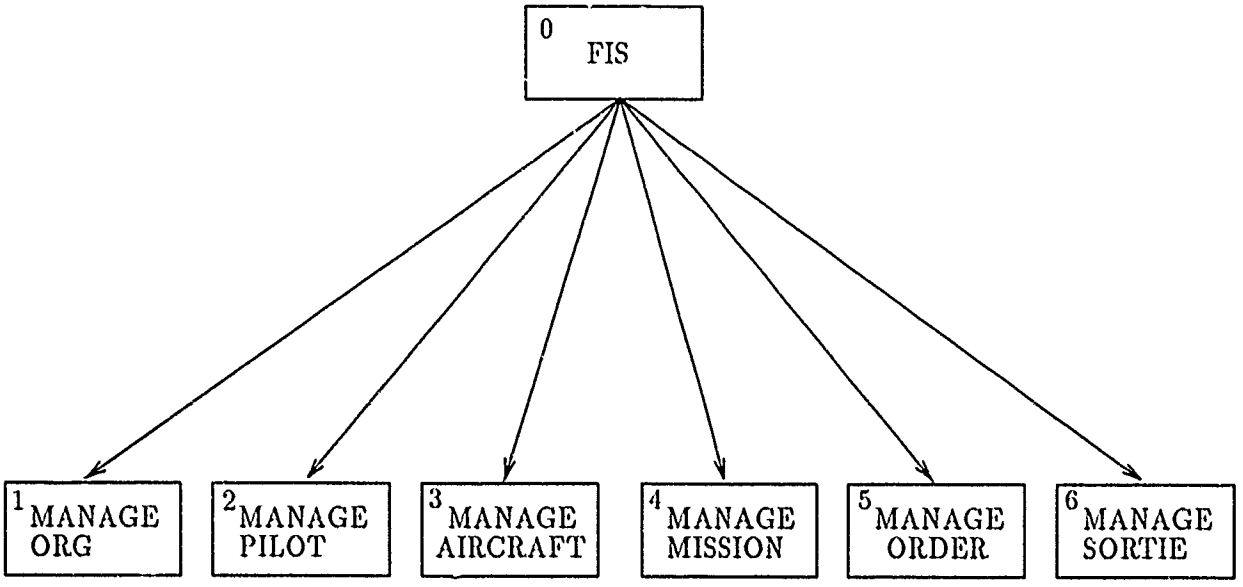


Figure 21. High-level Structure Chart of the FIS

4.3.2 Low-Level Design. The detailed DFDs of the FIS which are shown in Figure 8 through Figure 13 are transformed into the low-level structure chart of the FIS. Figure 22 through Figure 27 show the low-level structure chart of the FIS. Each of the modules of the low-level structure chart, with the exception of module 1.7.1, are transformed from a process of DFDs as shown in Figure 8 through Figure 13. Each of the low level modules will be developed using SQL*Forms or Pro*Ada.

Many modules such as 1.7, 2.4, 2.5, 3.3, and 6.3 need to read the information of *orgs* in a specific sequence. Module 1.7.1 is designed to support these modules. Module 1.7.1 stores *item (= org-code + org-name)* of specific *orgs* to a *circular-queue* and returns one at a time when it is called by higher level modules. The order of storing and returning *item* depends on the value of *sel* which is to be accepted from user.

The DFD of *organization* shown in Figure 8 is transformed into a structure chart as shown in Figure 22. Each of the processes are transformed into a module. Module 1.7 calls module 1.7.1 to receive specific *items* and then to display them. The functions of module 1.1 through module 1.7 and module 1.7.1 are described below.

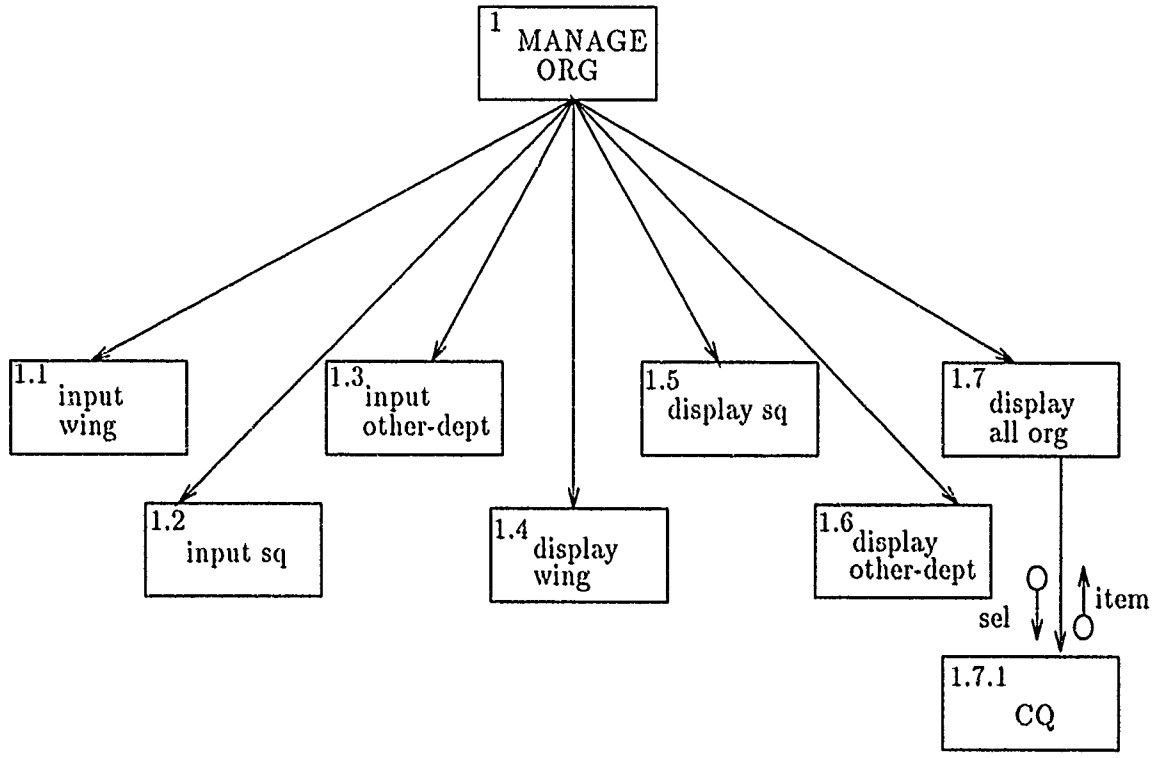


Figure 22. Structure Chart of Organization

Module 1.1

```
begin
repeat until user exit the program
      accept wing-code;
      if the wing-code exists in wing table
            then display wing-rec;
      end if;
      accept wing-rec;
      check validity of grand-msn-code;
      commit;
end repeat;
end;
```

Module 1.2

```
begin
repeat until user exit the program
    accept sq-code;
    if sq-code exists in sq table
        then display sq-rec;
    end if;
    accept sq-rec;
    check validity of wing-code;
    commit;
end repeat;
end;
```

Module 1.3

```
begin
repeat until user exit the program
    accept other-dept-code;
    if other-dept-code exists in other-dept table
        then display other-dept;
    end if;
    accept other-dept;
    commit;
end repeat;
end;
```

Module 1.4

```
begin
    display wing-rec from wing table order by grand-msn-code, wing-code;
end;
```

Module 1.5

```
begin
    display sq-rec from sq table order by wing-code, sq-code;
end;
```

Module 1.6

```
begin
    display other-dept from other-dept table order by other-dept-code;
end;
```

Module 1.7

```
begin
    call ADDORG of module 1.7.1 giving sel = 1;
    if circular queue is not empty then
    call POPORG of module 1.7.1 and display item;
    end if;
end;
```

Module 1.7.1

```
  define circular queue with100 items;
  item = {what-org + org-code + org-name + wing-code};
```

```
procedure ADDCQ        (CQ: in out queue;
                        IM: in item);
      add an item to the tail of the CQ and return it;
end ADDCQ;
procedure CLEARCQ      (CQ: in out queue);
      delete all items in the CQ;
end CLEARCQ;
function ISEMPTYCQ     (CQ: queue) return boolean;
      if CQ is empty then return true else return false;
end CLEARCQ;
procedure ADDORG       (CQ:in out queue;
                        SEL: in integer;
                        OCODE: in string);
select SEL from [1:2:3];
if SEL = 1 then
      add items to the CQ in the sequence of
      cac ⟶ {wing ⟶ {sq}} ⟶ {other-dept};
end if;
if SEL = 2 then
      add items to the CQ in the sequence of
      wing ⟶ {sq};
end if;
if SEL = 3 then
      add items to the CQ in the sequence of
      {sq};
end if;
end ADDORG;
procedure POPORG       (CQ: in out queue;
                        IM: out item);
      return one item by one item from CQ;
end POPORG;
```

The DFD of *pilot* shown in Figure 9 is transformed into a structure chart as shown in Figure 23. Each of the processes are transformed into a module. Module 2.4 and module 2.5 calls module 1.7.1. The functions of module 2.1 through module 2.5 are described below.

Module 2.1

```
begin
repeat until user exit the program
      accept rank-code;
      if the rank-code exists in rank table
```

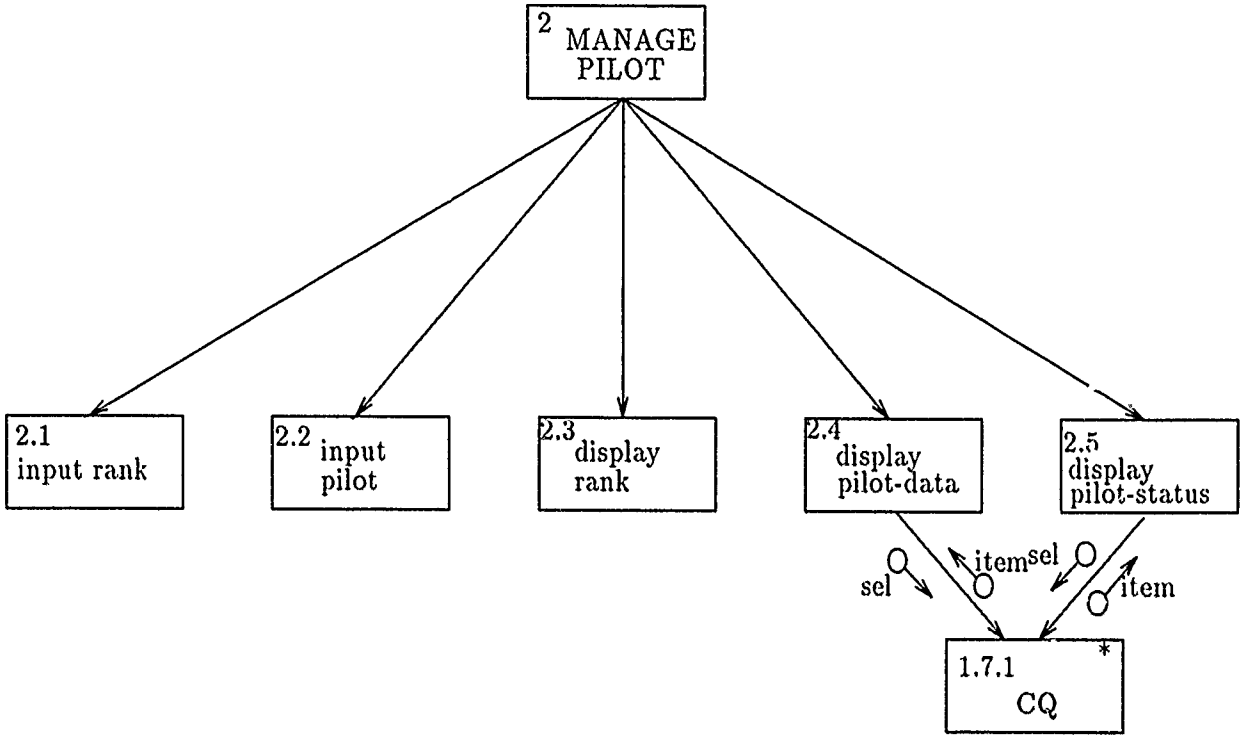


Figure 23. Structure Chart of Pilot

then display *rank*;
end if;
accept *rank*;
commit;
end repeat;
end;

Module 2.2

begin
repeat until user exit the program
accept *pilot-code*;
if *pilot-code* exists in *pilot* table
then display *pilot-rec*;
end if;
accept *pilot-rec*;
check validity of *ac-type* and *org-code*;
commit;
end repeat;
end;

Module 2.3

```
        begin
              display rank from rank table order by rank-code;
        end;
```

Module 2.4

```
        begin
        repeat until user exit the program
             select one from [all:wing:org];
             if selection = all then
                  display all pilot-rec from pilot table order by pilot-id;
                  else if selection = wing then
                       display all pilot-rec of the wing from pilot table order by pilot-id;
                       else if selection = org then
                            display all pilot-rec of the org from pilot table order by pilot-id;
             end if;
        end repeat;
        end;
```

Module 2.5

```
        begin
        repeat until user exit the program
             select one from [all:wing];
             if selection = all
                  then compute pilot-number of cac;
                  * pilot-number = pilot-tot+hospt+vact+off+ready *
                  display pilot-number of cac;
                  repeat until data end of wing table
                       compute pilot-number of the wing including their sqs;
                       display pilot-number of the wing and their sqs;
                  end repeat;
                  repeat until data end of other-dept table
                       compute pilot-number of the other-dept;
                       display pilot-number of the other-dept;
                  end repeat;
             end if;
             if selection = wing then
                  reapeat until data end of wing table
                       compute pilot-number of the wing;
                       display pilot-number of the wing;
                       repeat until data end of sq table
                            compute pilot-number of the sq;
                            display pilot-number of the sq;
                       end repeat;
                  end repeat;
             end if;
        end repeat;
        end;
```

The DFD of *aircraft* shown in Figure 10 is transformed into a structure chart as shown in Figure 24. Each of the processes are transformed into a module. Module 3.3 calls module 1.7.1. The functions of module 3.1 through module 3.5 are described below.

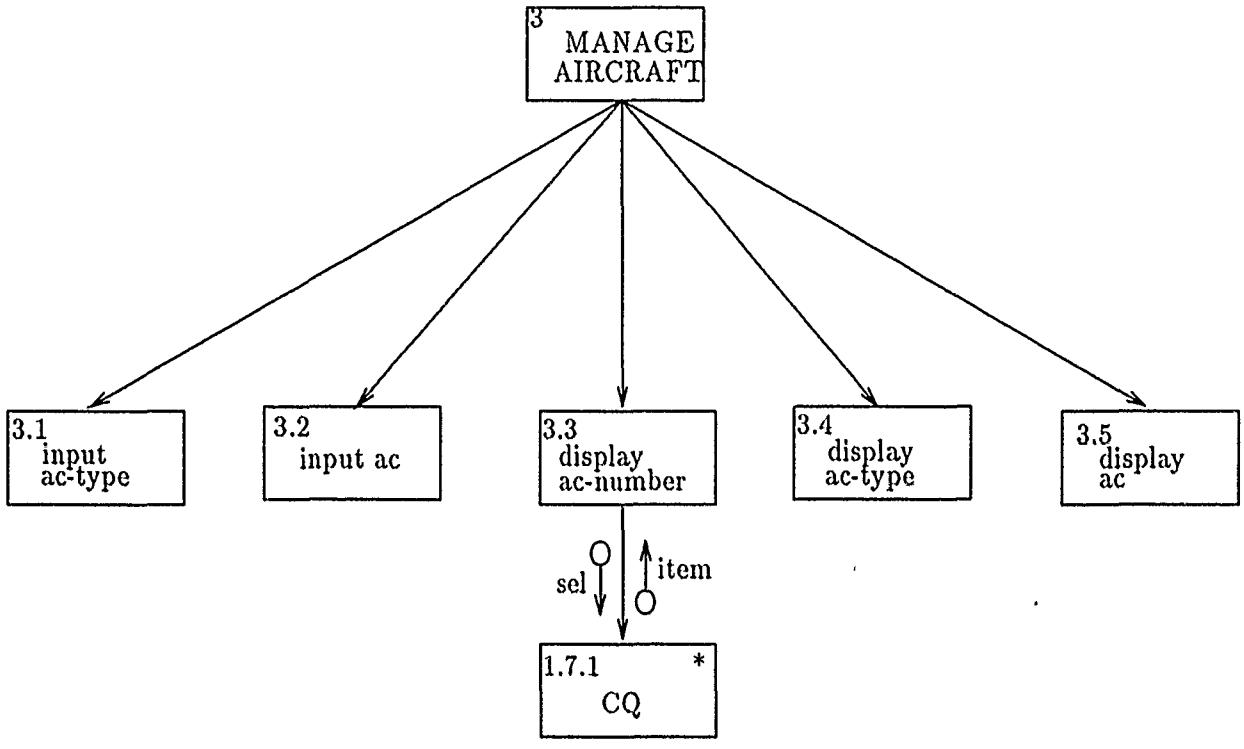


Figure 24. Structure Chart of Aircraft

Module 3.1

```
begin
repeat until user exit the program
        accept ac-type;
        if the ac-type exists in ac-type table
                then display ac-type-rec;
        end if;
        accept ac-type-rec;
        check validation of grand-msn-code;
        commit;
end repeat;
end;
```

Module 3.2

```
begin
repeat until user exit the program
        accept ac-no;
        if ac-no exists in ac table
                then display ac-rec;
        end if;
        accept ac-rec;
        check validation of ac-type, org-code;
        commit;
end repeat;
end;
```

Module 3.3

```
begin
repeat until user exit the program
        select from [all-ac-type:one-ac-type];
        if selection = all-ac-type then
                repeatuntil data end of ac-type table
                        compute ac-tot-status of the ac-type;
                        * ac-tot-status=ac-rdy+ac-maint+ac-tot+ac-pct *
                        display ac-tot-status of the ac-type;
                end repeat;
        end if;
        if selection = one-ac-type then
                accept ac-type;
                repeat until data end of org
                        compute ac-tot-status of the org, of the ac-type;
                        display ac-tot-status of the org, of the ac-type;
                end repeat;
        end if;
end repeat;
end;
```

Module 3.4

```
begin
        display all ac-type-rec from ac-type order by grand-msn-code, ac-type;
end;
```

Module 3.5

```
begin
        display ac-rec of ac table order by ac-no;
end;
```

The DFD of *mission* shown in Figure 11 is transformed into a structure chart as shown in Figure 25. Each of the processes are transformed into a module. The functions of module 4.1 through module 4.6 are described below.

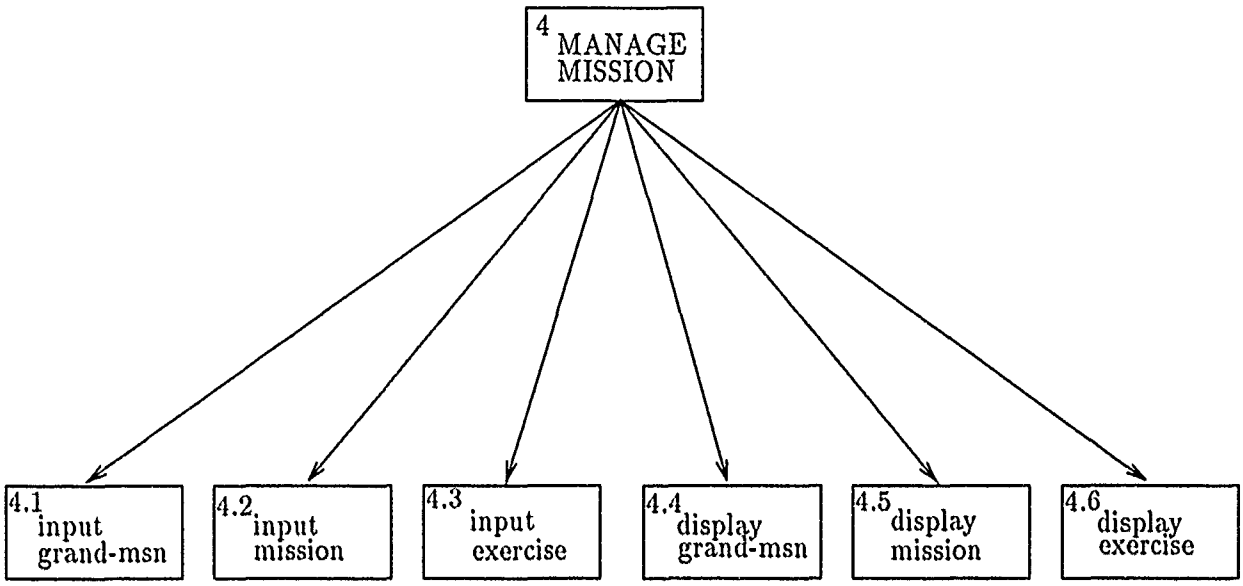


Figure 25. Structure Chart of Mission

Module 4.1

```
begin
repeat until user exit the program
        accept grand-msn-code;
        if the grand-msn-code exists in grand-msn table;
                then display grand-msn;
        end if;
        accept grand-msn;
        commit;
end repeat;
end;
```

Module 4.2

```
begin
repeat until user exit the program
        accept msn-code;
        if msn-code exists in mission table
                then display msn-rec;
        end if;
        accept msn-rec;
        check validity of grand-msn-code;
        commit;
end repeat;
end;
```

Module 4.3

```
begin
repeat until user exit the program
        accept ex-code;
        if ex-code exists in exercise table
                then display exercise;
        end if;
        accept exercise;
        commit;
end repeat;
end;
```

Module 4.4

```
begin
        display grand-msn of grand-msn table order by grand-msn-code;
end;
```

Module 4.5

```
begin
        display mission-rec of mission table order by msn-code;
end;
```

Module 4.6

```
begin
        display exercise of exercise table order by ex-code;
end;
```

The DFD of *flight-order* shown in Figure 12 is transformed into a structure chart as shown in Figure 26. Each of the processes are transformed into a module. The functions of module 5.1 through module 5.4 are described below.

Module 5.1

```
begin
repeat until user exit the program
        accept cac-order-no;
        if the cac-order-no exists in cac-order table;
                then display cac-order-rec;
        end if;
        accept cac-order-rec;
        check validity of ac-type, ex-code, msn-code, and wing-code;
        update change-time of data-change table to current time;
        commit;
end repeat;
end;
```

Module 5.2

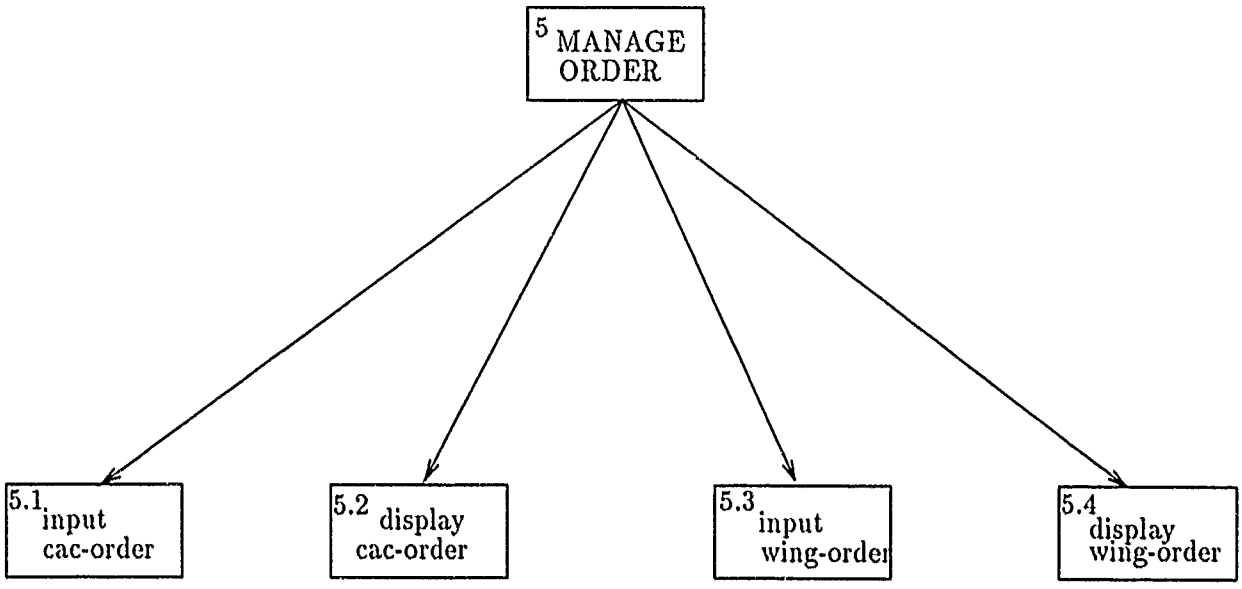


Figure 26. Structure Chart of Flight-Order

```
        begin
        repeat until user exit the program
                accept wing-code + wing-order-no;
                if wing-code + wing-order-no exists in wing-order table;
                        then display wing-order-rec;
                end if;
                accept wing-order-rec;
                check validity of ac-type, ex-code, msn-code, sq-code, and wing-code;
                update change-time of data-change table to current time;
                commit;
        end repeat;
        end;
```

Module 5.3

```
        begin
        time := 0;
        task get-command is
                loop
                        accept command;
                end loop;
        end task;
        task main-prog is
                loop
                        if command = '999'
```

```
                    then exit this loop;
                end if;
                wing-code := command;
                read change-time of the wing-code from data-change table;
                if change – time > time or wing-code changed
                    then display cac-order of the wing-code from cac-order table;
                end if;
                time := change-time;
            end loop;
            terminate task get-command;
            end task;
        end;
```

Module 5.4

```
        begin
        time := 0;
        task get-command is
            loop
                accept command;
            end loop;
        end task;
        task main-prog is
            loop
                if command = '999'
                    then exit this loop;
                end if;
                org-code := command;
                read change-time of the org-code from data-change table;
                if change – time > time or org-code changed
                    then display wing-order of the org-code from cac-order table;
                end if;
                time := change-time;
            end loop;
            terminate task get-command;
            end task;
        end;
```

The DFD of *flight-sortie* shown in Figure 13 is transformed into a structure chart as shown in Figure 27. Each of the processes are transformed into a module. Module 6.3 calls module 1.7.1. The functions of module 6.1 through module 6.7 are described below.

Module 6.1

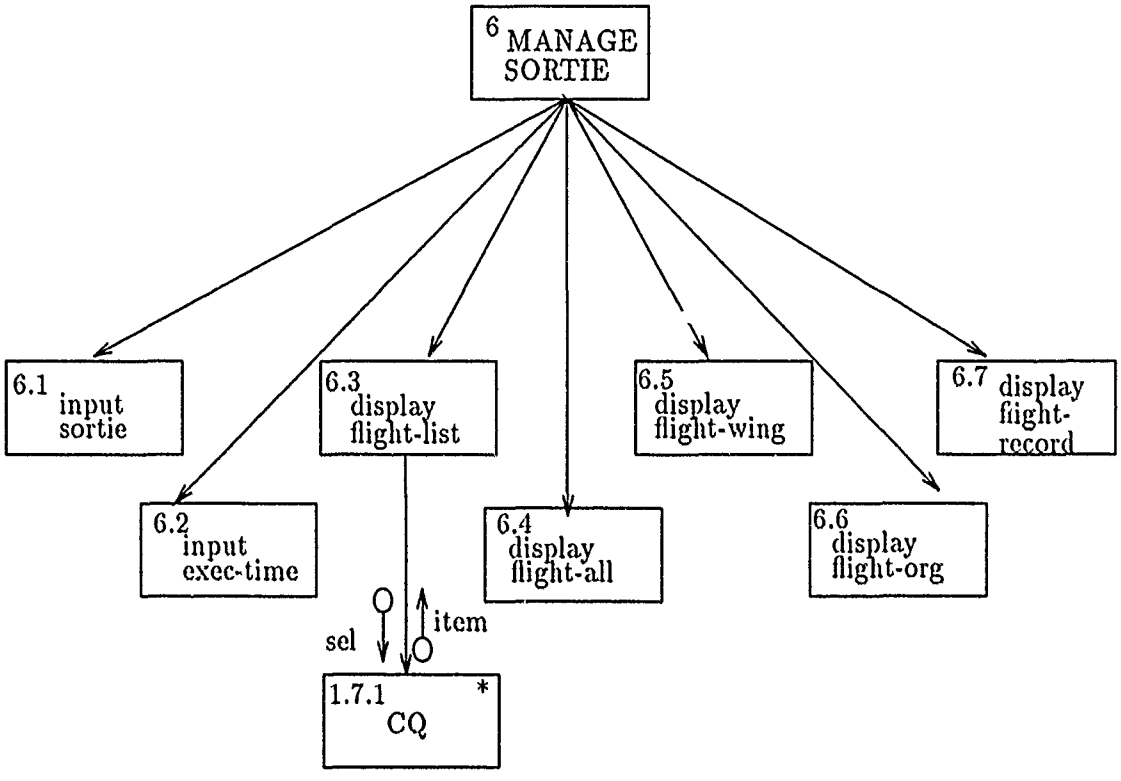


Figure 27. Structure Chart of Flight-Sortie

begin
repeat until user exit the program;
accept *plan-take-off + sq-code + set-no*;
if the *plan-take-off+sq-code+set-no* exists in *set-sortie* table
then display *set-sortie-rec*;
end if;
accept *set-sortie-rec*;
check validity of *ac-type, ex-code, msn-code*, and *org-code*;
update *change-time* of *data-change* table to current time;
commit;
repeat until user exit the routine
accept *plan-take-off+sq-code+set-no+position-no*;
if the plan-take-off+sq-code+set-no+position-no exists in *sortie* table
then display sortie-rec;
end if;
accept sortie-rec;
check validity of *ac-no* and *pilot-id*;
update *change-time* of *data-change* table to current time;
commit;

```
        end repeat;
    end repeat;
    end;
```

Module 6.2

```
    begin
    repeat until user exit the program
        update take-off or update landing of sortie table;
        update change-time of data-change table;
        commit;
    end repeat;
    end;
```

Module 6.3

```
    begin
    time := 0;
    task get-command is
        loop
            accept command;
        end loop;
    end task;
    task main-prog is
        loop
            if command = '999'
                then exit this loop;
            end if;
        end loop;
        loop
            read MAX(change-time) of data-change table;
            if change - time > time then
                repeat until data end of wing table
                    display wing-code of wing table;
                    repeat until data end of sq table
                        display sq-code of sq table;
                        compute flight-status-list of the sq;
                        display flight-status-list of the sq;
                    end repeat;
                end repeat;
                repeat until data end of other-dept table
                    if sort - tot ≠ 0 then
                        compute flight-status-list of the other-dept;
                        display flight-status-list of the other-dept;
                    end if;
                end repeat
            end if;
            time := change-time;
        end loop;
```

terminate task *get-command*;
end;

Module 6.4

begin
display *set-sorties* from *set-sortie* table order by *plan-take-off, org-code, set-no*;
display *sorties* from *sortie* table order by *plan-take-off, org-code, set-no, position-no*;
end;

Module 6.5

begin
accept *wing-code*;
display *set-sorties* of the *wing-code* from *set-sortie* table order by *plan-take-off, org-code, set-no*;
display *sorties* of the *wing-code* from *sortie* table order by *plan-take-off, org-code, set-no, position-no*;
end;

Module 6.6

begin
accept *org-code*
display *set-sorties* of the *org-code* from *set-sortie* table order by *plan-take-off, org-code, set-no*;
display *sorties* of the *org-code* from *sortie* table order by *plan-take-off, org-code, set-no, position-no*;
end

Module 6.7

begin
accept selections (*from-date, to-date, org-code*, and/or *pilot-id*);
display *sorties* which satisfy the selections from *sortie* table order by *take-off*;
end;

V. Implementation and Testing

This chapter presents the process of implementation and testing of the FIS. A top-down development technique was applied in this process. The implementation and the test were done simultaneously. Each module of the structure chart was implemented and tested. The top modules were implemented before the bottom modules and the left-side modules were implemented before the right-side modules. The functions and procedures of each module were derived from the module descriptions generated in Chapter 4.

Whenever a module was implemented, two testing steps were followed. The first step was the test of the module itself. In this step, each function and constraint of the written program was compared with the module description generated in Chapter 4 by inserting or displaying sample data. The second step was the connection test of each module between the modules with which data items were exchanged. This step of the test included the entire modules implemented previously. The modules being tested were called by other modules to verify a connection exists between them.

SQL*Menu, SQL*Forms, and Pro*Ada were used to implement the FIS. The process of implementation and testing of the FIS can be grouped into four steps each of which are presented below.

The first step was the implementation and test of the high-level structure chart. Each module of the high-level structure chart does not change data items. The only required function of each module is calling of its sub-modules. The high-level structure chart consisting of module 0, 1, 2, 3, 4, 5, and 6 were implemented and tested using SQL*Menu. Appendix C shows the high-level of the FIS shown on the screen.

The second step was the implementation and test of the module 1.7.1. Though module 1.7.1 is the lowest level module, it was developed earlier than some of other higher level modules because it is called by many higher level modules. By implementing it earlier, each module which calls module 1.7.1 can be tested with a real sub-program. Pro*Ada was used as the programming language and *Circular-Queue* data structure[1] was used. Module 1.7.1 consists of several functions/procedures

[1]This *Circular-Queue* has a capacity of 100 items which can add or pop the information related to an org.

and it returns information of *org* one at a time. The sequence of returning information depends the user input as shown in Module 1.7.1.

The third step was the implementation and test of each of the sub-modules of module 1 through module 6. Pro*Ada and SQL*Forms were used in this step. The modules which require complex procedures such as modules 1.7, 2.4, 2.5, 3.3, 3.5, 5.3, 5.4, and 6.3 were implemented using Pro*Ada. And the rest of the modules were implemented using SQL*Forms.

Modules 5.3, 5.4, and 6.3 required peculiar data manipulation procedures. Many users may input data through four kinds of input programs (Modules 5.1, 5.2, 6.1, and 6.2) and many users may display them at the same time. Modules 5.3, 5.4, and 6.3 require the data to be displayed simultaneously on the screen as the user insert it. *Data-change* is an intermediate table between input programs and output programs (Modules 5.3, 5.4, and 6.3). *Data-change* table keeps each *org*'s *change-time* which contains newest change time of data and each input programs update the *change-time* of a certain *org* into current time. Then three output programs display newest data continuously by comparing latest display-time with the *change-time*. Pro*Ada's *task*s were used to implement this problem. Each of the three modules, 5.3, 5.4, 6.3, consists of a main program and a *task*. The task accepts commands from the user keyboard and the main program responds to it. The following program shows how the *task* was used to implement this problem.

```
1  with ...;              use ...;
2  procedure p53 is
   ..............
3  change-time       : string(1..20) := (1..20 => '0');
4  pre-change-time   : string(1..20) := (1..20 => '0');
5  q                 : cq.queue;
6  command           : cq.item;
7  outfile           : file-mode;
8  out-file          : file-type;
9  task get-command;
10 task body get-command is
11 begin
12 loop
13    get(command.org-code);
14    CQ.ADDCQ(q, command);
15    if command.org-code = "999" then exit; end if;
16 end loop;
```

```
17 end get-command;
18 begin
      .....
19 loop
20   getcom: loop
21      if not CQ.ISEMPTYCQ(q) then
22         CQ.POPORG(q, command);
23         exit when command.org-code = "999";
24         org-code := command.org-code;
25      end if;
26      EXEC SQL SELECT to-char(change-time,'DD-MON-YYYY HH24:MI:SS')
27      INTO :change-time
28      FROM data-change
29      WHERE org-code = :org-code and change-code = :change-code;
30      exit when change-time /= pre-change-time;
31      TEXT-IO.OPEN(out-file, outfile, "[ykwak.thesis]dummy-file.");
32      TEXT-IO.CLOSE(out-file);
33   end loop getcom;
34   if command.org-code = "999" then exit; end if;
        ..... DISPLAY FLIGHT-ORDER .....
35   pre-change-time := change-time;
36 end loop;
        .....
37 end p53;
```

The final step was the testing of the FIS. Before this test, all data in the database tables were cleared to insert and display new sample data. Insertion of data into a database table required reference to data of another table to validate it. Table 23 shows the database tables which should be referred to when inserting data into a database table. For example, the *sq* table must refer to the *wing* table and the *wing* table must refer to the *grand-msn* table. Input programs were tested by inserting sample data in the following sequence:

1. grand-msn, rank, exercise, other-dept, data-change
2. wing, mission, ac-type
3. sq, cac-order
4. pilot, ac, wing-order, set-sortie
5. sortie

No	Input Table	Reference Table(s)
1	ac	ac-type, sq, other-dept
2	ac-type	grand-msn
3	cac-order	wing, mission, ac-type, exercise
4	data-change	
5	exercise	
6	grand-msn	
7	mission	grand-msn
8	other-dept	
9	pilot	sq, other-dept, rank
10	rank	
11	set-sortie	ac-type, exercise, mission, sq, other-dept
12	sortie	set-sortie, ac, pilot
13	sq	wing
14	wing	grand-msn
15	wing-order	ac-type, exercise, mission, sq

Table 23. Reference Tables for Data Input

Output programs were tested by displaying sample data inserted previously.

VI. Conclusion and Recommendation

This chapter summarizes the work accomplished in this thesis. Also, it presents the conclusions of this thesis work. Finally, it recommends a better way for FIS development and the better use of the software products.

6.1 Summary

The goal of this thesis, the development of a FIS database system, was successfully accomplished. The FIS was developed using the structured methods and the ORACLE RDBMS through the following three steps.

The first step was the analysis of the FIS. In this step, the requirements of the users were identified and the perception of the real world of the FIS was modeled into a database structure. The structured analysis technique using tools such as DFD, DD, process specification, and ERD were used.

The second step was the design of the FIS. ORACLE database tables were generated from DD using the relational scheme normalization technique. Also, a structure chart with a module description for each of the modules was constructed.

The third step was the implementation and testing of the FIS. Each module of the structure chart were implemented and tested one by one, from top to bottom. The high-level modules of the structure chart were implemented using SQL*Menu while the rest of the modules were implemented using SQL*Forms or Pro*Ada. Finally, the whole system was tested by inserting and displaying sample data.

6.2 Conclusion

This thesis work employed the structured method using a variety of tools as well as ORACLE which is expected to be the most popular database system in Korea. Several kinds of the ORACLE products, available, SQL*Menu, SQL*Forms, and Pro*Ada, were used in this thesis. The following paragraphs summarizes the conclusions of this thesis work.

ORACLE is useful for the development of data processing systems. Much development time could be saved by using ORACLE products such as SQL*Menu and SQL*Forms. For example, SQL*Forms could save approximately 50% of the coding time compared with Pro*Ada, a high-level programming language. Also, several of the ORACLE product allows easy to develop and easy to change. For example, SQL*Menu can be used to combine sub-programs into a menu and SQL*Forms can design the input or output format of the program quickly. Once developed, the programs can be updated continuously as required. This allows the programmer to develop a proto-type of a system easily.

The module descriptions of the structure chart are important especially when they are implemented with SQL*Forms. The programming style of SQL*Forms is different from that of other high-level programming languages such as Ada, C, Cobol, etc. While it is easy to develop a program using SQL*Forms it is difficult to read the written program. It is easier to read a program written in a high-level language compared with one written using SQL*Forms. There are two ways of reading a program written in the SQL*Forms. One is by reading an .INP file which is generated by SQL*Forms automatically. Because the .INP file is a dialogue type program and the size of it is much larger than that of a program written in a high-level language, it is difficult to understand it. The other way is by tracing each BLOCK and FIELD of each FORM of the program on the screen using SQL*Forms. This also is difficult to understand since one need to read many screens to understand the whole program and it is hard to remember the contents of the previous screens. Thus, the module specification need to be specified the functions of the program in structured detail format. By reading the module description instead of the program, one can understand easily and save time.

6.3 Recommendation

This thesis work does not provide the best solution to the questions *"What to develop?"* and *"How to develop?"* the FIS. This thesis work limited the scope of the FIS and kept the software engineering life cycle on track using the selected methodology and tools. Based on this thesis work, the following recommendations are proposed to better develop the FIS or any other database system in the future.

This thesis work assumed that all users use only one computer located at the CAC with one or more terminal(s) connected to the computer. The scope of the real FIS includes many sites and each of them have a mainframe connected to another sites's mainframe through computer network. Also, the real FIS requires more than 150 terminals, some of them are located at great distances from the mainframe. The FIS implemented with only one mainframe cannot support the users satisfactorily. This situation forces the use of *distributed database system* techniques. SQL*Net is an ORACLE product which supports distributed database system (11).

> Oracle's SQL*Net network software and the ORACLE RDBMS make data distributed over multiple, incompatible networks and computers appear as a single database on a single computer. With SQL*Net, you can integrate diverse hardware, operating systems, databases, communications protocols, and applications to create a unified computing information resource.

With a distributed database system, the load on the mainframe of the CAC can be decreased. Distributed database systems, however, require a highly reliable computer network between cites.

In this thesis, the module description for each module of the structure chart did not consider the characteristics of programming in SQL*Forms. As a result, the module descriptions are not useful enough for programming and maintaining the programs. As discussed in Section 6.2, the modules of a structure chart implemented with SQL*Forms need to be well-described. An organized and unified module description form is needed to provide a well described module description.

By using ORACLE products, much of the development time could be saved. This thesis applied five kinds of ORACLE products. As of 1991, ORACLE Corporation provides a variety of ORACLE products as shown below (19):

1. Database and Networking
   - ORACLE Relational DBMS N6.0
   - SQL*Net
   - SQL*Connect to DB2 and SQL/DS
   - SQL*Connect to TurboIMAGE

2. CASE and Application Development Tools
   - CASE*Method

- CASE*Dictionary
- CASE*Designer
- CASE*Generator
- PL/SQL
- The ORACLE Precompilers
- SQL*Forms V3.0
- SQL*Menu V5.0
- SQL*ReportWriter
- ORACLE Graphics
- SQL*Plus
- SQL*TextRetrieval

3. Office Automation and End-User Tools
   - Oracle*Mail
   - SQL*QMX
   - ORACLE database add-in for Lotus 1-2-3
   - ORACLE for 1-2-3 DataLens
   - Easy*SQL
   - SQL*Calc

Through careful selection of the ORACLE products, one can save much software development time and can generate a good quality software system.

Appendix A. *Data Dictionary*

DATA	DESCRIPTION
ac	= ac-no + ac-status + start-date
ac-maint	= *number of aircrafts which are on maintenance*
ac-no	= *an unique number given to an aircraft which is running*
ac-pct	= ac-rdy / ac-tot * 100
ac-rdy	= *number of aircrafts which are ready to take-off*
ac-rec	= {ac-no + ac-type + ac-code + ac-status + start-date}
ac-status	= [R:M] *status of an aircraft R = ready to take-off, M = on maintenance*
ac-tot	= *total number of aircrafts* ac-rdy + ac-maint
ac-tot-status	= *statistic data of aircrafts status* {ac-rdy + ac-maint + ac-tot + ac-pct}
ac-type	= *name of an aircraft kind which is running in the Air Force*
ac-type-rec	{ac-type + grand-msn-name}
blood-type	= [A:B:O:AB]
cac-order	= *flight-orders from CAC to each wing* {cac-order-no + number-ac + take-off + cac-descript}
cac-descript	= *description about a flight order of cac*
cac-order-no	= *unique number of a cac-order*
cac-order-rec	={cac-order-no + wing-code + take-off + msn-code + ex-code + ac-type + number-ac + cac-descript}
change-code	= [OC:OW:SC:SE] *OC = cac-order added or changed, OW = wing-order added or changed, SC = flight-plan added or changed, SE = aircraft took-off or landed*
change-time	= *change-code updated time*
class	= *class of pilots which denotes the flight capability of them* [IP:FC:EC:WM:ST]
data-change	= *this data is used to display some important data simultaneously as it changed* {org-code + change-code + change-time}
date	= [year + month + day : day + month + year]
exec-time	= [take-off : landing]
exercise	=*military exercise of Air Force which require aircraftS* {ex-code + ex-name}

ex-code	= *code of an exercise of Air Force*
ex-name	= *name of an exercise of Air Force*
flight-status	= [R:O:F] *this data is used to denote the status of a sortie* R=ready, O=on air, F=finish*
flight-status-all	={plan-take-off-d+plan-take-off-t+org-code+set-no+msn-code + ex-code+ac-type+1{position-no+ac-no+take-off-d+take-off-t+ landing-d+landing-t+front-pilot+rear-pilot+stand-by-pilot+ flight-status}4}
flight-status-list	= {org-code + sort-tot + sort-rdy + sort-air + sort-fin + pct-rdy + pct-air + pct-fin}
flight-status-org	= org-code + flight-status-all
flight-status-wing	= wing-code + flight-status-all
front-pilot	= *pilot-id of front-seat or main seat pilot*
grand-msn	= *grand mission of Air Force assigned to a wing or a ac-type* {grand-msn-code + grand-msn-name}
grand-msn-code	= [A:C:O:T] *A=fight, C=carry, O=observe, T=training*
grand-msn-name	= *name of a grand-msn-code*
hospt	= *total number of pilots in hospital*
job	= *a job name of a pilot*
landing	= *landing time* date + time
maint	= *total number of aircrafts on maintenance*
mission	= *a flight mission assigned to a set of sortie* msn-code + msn-name
msn-code	= *a code of an Air Force flight mission*
msn-name	= *name of an Air Force flight mission*
msn-rec	= {msn-code + msn-name + grand-msn-code}
name	= *name of a pilot* first name + , + initial of last name
number-ac	= *number of aircraft required by a cac-order or a wing-order*
off	*total number of pilots on duty off*
org-all	={org-code + org-name}
org-code	= [CAC:wing-code:sq-code:other-dept-code]
other-dept	= *departments where pilots stay except CAC, wing and sq* {other-dept-code + other-dept-name}
other-dept-code	*code of an other-dept*
other-dept-name	*name of an other-dept*
pilot	= *a person who is committed to control aircraft of Air Force* pilot-id + name + class + blood-type + pilot-date + job + pilot-status
pilot-date	= *pilot qualified date*

pilot-id	= *an unique number given to a pilot* *range= 10,000 - 99,999*
pilot-list1	= {org-code + pilot-id + name + rank-code + ac-type + pilot-status}
pilot-list2	= {org-code + pilot-tot + hospt + vact + off + ready}
pilot-rec	={pilot-id + name + rank-code + ac-type + class + blood-type + pilot-date + job + pilot-status}
pilot-status	= [R:H:O:V] *status of a pilot R=ready to take off, H=in hospital, O=duty off, V=on vacation*
pilot-tot	= *total number of pilot*
plan-take-off	= *take-off time of a flight-plan*
position-no	= *position number of an aircraft in a set of sortie*
rank	= *military grade of Air Force officer* {rank-code + rank-name}
rank-code	= *code of rank*
rank-name	= *name of rank*
ready	= *total number of pilots who are ready to take-off*
rear-pilot	= *pilot-id of a rear seated pilot*
run	= *total number of aircrafts that are ready to take-off*
set-no	= *sequential number of set of flight plan*
set-sortie	= *a set of flight sortie which consist one to four sortie* set-no + plan-take-off
set-sortie-rec	={plan-take-off + org-code + set-no + msn-code + ex-code + ac-type}
sortie	= *a flight action between take off to landing by one to three pilot(s) with an aircraft* position-no + take-off + landing
sortie-rec	={plan-take-off + org-code + set-no + position-no + ac-no + take-off + landing + front-pilot + rear-pilot + stand-by-pilot + flight-status
sq	= *a flight squadron of Air Force* sq-code + sq-name + sq-establish
sq-code	= *squadron code*
sq-establish	= *squadron established date*
sq-name	= *squadron name*
sq-rec	= {wing-code + sq-code + sq-name + sq-establish
stand-by-pilot	= *pilot-id of a stand-by-pilot in a carrier ac*
start-date	= *start date to run of an aircraft*
take-off	= *take off time of an aircraft* date + time
time	= hour + minute + [second]
vact	=*total number of pilots who are on vacation*

wing	= *flight wing of Air Force* wing-code + wing-name + wing-establish
wing-code	= *wing code*
wing-descript	= *description of flight order from wing*
wing-establish	= *wing established date*
wing-rec	= {grand-msn-code + wing-code + wing-name + wing-establish}
wing-name	= *name of a wing*
wing-order	= *flight-order from wing to each of their squadron* wing-order-no + take-off + wing-descript
wing-order-no	= *sequential number of flight-order from wing*
wing-order-rec	={org-code + wing-order-no + take-off + msn-code + ex-code + ac-type + number-ac + wing-descript}

Appendix B. *Database Table and Index Creation Program*

```
create table ac          (ac-no          number(4)    not null,
                          ac-type        char(6)      not null,
                          org-code       char(3)      not null,
                          ac-status      char(1)      not null,
                          start-date     date         not null);

create table ac-type     (ac-type        char(6)      not null,
                          grand-msn-code char(1)      not null);

create table cac-order   (cac-order-no   number(3)    not null,
                          wing-code      char(3)      not null,
                          take-off-d     date,
                          take-off-t     number(4),
                          msn-code       char(2)      not null,
                          ex-code        char(2),
                          ac-type        char(6)      not null,
                          number-ac      number(1)    not null,
                          cac-descript   char(200));

create table data-change (org-code       char(3)      not null,
                          change-code    char(2)      not null,
                          change-time    date         not null);

create table exercise    (ex-code        char(2)      not null,
                          ex-name        char(15)     not null);

create table grand-msn   (grand-msn-code char(1)      not null,
                          grand-msn-name char(10)     not null);

create table mission     (msn-code       char(2)      not null,
                          msn-name       char(10)     not null,
                          grand-msn-code char(1)      not null);

create table other-dept  (other-dept-code char(3)     not null,
                          other-dept-name char(16)    not null);

create table pilot       (pilot-id       number(5)    not null,
                          name           char(10)     not null,
                          rank-code      char(2)      not null,
                          org-code       char(3)      not null,
                          ac-type        char(6)      not null,
```

```
                    class           char(2)     not null,
                    blood-type      char(2)     not null,
                    pilot-date      date,
                    job             char(10),
                    pilot-status    char(1)     not null);

create table rank   (rank-code      char(2)     not null,
                    rank-name       char(10)    not null);

create table set-sortie (plan-take-off-d date   not null,
                    plan-take-off-t number(4)   not null,
                    org-code        char(3)     not null,
                    set-no          number(2)   not null,
                    msn-code        char(2)     not null,
                    ex-code         char(2),
                    ac-type         char(6)     not null,
                    number-ac       number(1)   not null);

create table sortie (plan-take-off-d date       not null,
                    plan-take-off-t number(4)   not null,
                    org-code        char(3)     not null,
                    set-no          number(2)   not null,
                    position-no     number(1)   not null,
                    ac-no           number(4)   not null,
                    take-off-d      date,
                    take-off-t      number(4),
                    landing-d       date,
                    landing-t       number(4),
                    front-pilot     number(5)   not null,
                    rear-pilot      number(5),
                    stand-by-pilot  number(5),
                    flight-status   char(1)     not null);

create table sq     (sq-code        char(3)     not null,
                    sq-name         char(10)    not null,
                    sq-establish    date        not null,
                    wing-code       char(3)     not null);

create table wing   (wing-code      char(3)     not null,
                    wing-name       char(10)    not null,
                    wing-establish  date        not null,
                    grand-msn-code  char(1)     not null);

create table wing-order (sq-code    char(3)     not null,
                    wing-order-no   number(2)   not null,
```

```
take-off-d       date,
take-off-t       number(4),
msn-code         char(2)      not null,
ex-code          char(2)      not null,
ac-type          char(6)      not null,
number-ac        number(1)    not null,
wing-descript    char(200));
```

```
create unique index ac-i              on ac(ac-no);
create unique index ac-type-i         on ac-type(ac-type);
create unique index cac-order-i       on cac-order(cac-order-no);
create unique index data-change-i     on data-change(org-code, change-code);
create unique index exercise-i        on exercise(ex-code);
create unique index grand-msn-i       on grand-msn(grand-msn-code);
create unique index mission-i         on mission(msn-code);
create unique index other-dept-i      on other-dept(other-dept-code);
create unique index pilot-i           on pilot(pilot-id);
create unique index rank-i            on rank(rank-code);
create unique index set-sortie-i      on set-sortie(plan-take-off-d, plan-take-off-t, org-code,
                                      set-no);
create unique index sortie-i          on sortie(plan-take-off-d, plan-take-off-t, org-code, set-no,
                                      position-no);
create unique index sq-i              on sq(sq-code);
create unique index wing-i            on wing(wing-code);
create unique index wing-order-i      on wing-order(sq-code, wing-order-no);
```

Appendix C. *Screen Design of the High-Level FIS*

```
<< MODULE 0 >>

+-------------------------------------------------------------+
|                  Flight Information System                  |
|                         Main Menu                           |
|                                                             |
|                  1  Manage Organization                     |
|                  2  Manage Pilot                            |
|                  3  Manage Aircraft                         |
|                  4  Manage Flight Mission                   |
|                  5  Manage Flight Order                     |
|                  6  Manage Flight Sortie                    |
|                                                             |
+-------------------------------------------------------------+


<< MODULE 1 >>

+-------------------------------------------------------------+
|                    1. Manage Organization                   |
|           ---------------------------------------           |
|                                                             |
|                    1  Input Wing                            |
|                    2  Input Squadron                        |
|                    3  Input Other Department                |
|                    4  Display Wing                          |
|                    5  Display Squadron                      |
|                    6  Display Other Department              |
|                    7  Display All Organization              |
|                                                             |
+-------------------------------------------------------------+
```

<< MODULE 2 >>

```
+-------------------------------------------------------------+
|                      2. Manage Pilot                        |
|           ---------------------------------------           |
|                                                             |
|              1  Input Rank                                  |
|              2  Input Pilot                                 |
|              3  Display Rank                                |
|              4  Display Personal Information of Pilot       |
|              5  Display Statistic Number of Pilot           |
|                                                             |
|                                                             |
|                                                             |
+-------------------------------------------------------------+
```

<< MODULE 3 >>

```
+-------------------------------------------------------------+
|                      3. Manage Aircraft                     |
|           ---------------------------------------           |
|                                                             |
|              1  Input Aircraft Type                         |
|              2  Input Aircraft                              |
|              3  Display Statistic Numbers of Aircraft       |
|              4  Display Aircraft Type                       |
|              5  Display Detail Information of Aircraft      |
|                                                             |
|                                                             |
+-------------------------------------------------------------+
```

<< MODULE 4 >>

```
+-------------------------------------------------------------+
|                      4. Manage Mission                      |
|           ---------------------------------------           |
|                                                             |
|                   1  Input Grand Mission                    |
|                   2  Input Mission                          |
|                   3  Input Exercise                         |
|                   4  Display Grand Mission                  |
|                   5  Display Mission                        |
|                   6  Display Exercise                       |
|                                                             |
+-------------------------------------------------------------+
```

<< MODULE 5 >>

```
+-------------------------------------------------------------+
|                    5. Manage Flight Order                   |
|           ---------------------------------------           |
|                                                             |
|                1  Input/Delete CAC Flight Order             |
|                2  Input/Delete Wing Flight Order            |
|                3  Display CAC Flight Order                  |
|                4  Display Wing Flight Order                 |
|                                                             |
|                                                             |
+-------------------------------------------------------------+
```

<< MODULE 6 >>

```
+-------------------------------------------------------------+
|                  6. Manage Flight Sortie                    |
|          ---------------------------------------            |
|                                                             |
|        1  Input Flight Sortie                               |
|        2  Input Flight Execution Time                       |
|        3  Display Statistic Number of Sortie                |
|        4  Display Flight Status of All Organization         |
|        5  Display Flight Status of a Wing                   |
|        6  Display Flight Status of an Organization          |
|        7  Display Flight Record                             |
|                                                             |
+-------------------------------------------------------------+
```

Bibliography

1. Adnan Altunisik. *Transfering 4GL Applications from Ingres to Oracle.* MS thesis, AFIT/GCE/ENG/91M-01. School of Engineering, Air Force Institute of Technology(AU), Wright Patterson AFB OH, 1991.
2. Antonio F. Silva. *Document Control and Retrieval System for the Brazilian Air Force.* MS thesis, AFIT/GCS/ENG/89J-2. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB OH, 1989.
3. Department of Defense. *Ada Programming Language.* Military Standard, ANSI/MIL-STD-1815A. 1980.
4. Edward Yourdon. *Managing The Structured Techniques.* New York: Yourdon Press, 1979.
5. Edward Yourdon. *Managing The System Life Cycle.* Englewood Cliffs, NJ: Yourdon Press, 1988.
6. Edward Yourdon. *Modern Structured Analysis.* Englewood Cliffs, NJ: Yourdon Press, 1989.
7. Henry F. Korth. *Database System Concepts.* New York: McGraw-Hill Book Company, 1986.
8. James Martin. *Fourth Generation Languages.* Englewood Cliffs, NJ: Prentice-Hall, 1986.
9. Oracle Corporation *Database Administrator's Guide. Version 6.0.* 1988.
10. Oracle Corporation. *Error Message and Codes Manual. Version 6.0.* 1988.
11. Oracle Corporation. *Guide to Oracle Products.* 1991.
12. Oracle Corporation. *Pro*Ada Precompiler User's Guide. Version 1.2.* 1987.
13. Oracle Corporation. *Pro*Ada User's Guide. Version 1.1.* 1986.
14. Oracle Corporation. *SQL*Forms Designer's Reference Manual. Version 2.0.* 1988.
15. Oracle Corporation. *SQL*Forms Designer's Tutorial. Version 2.3.* 1987.
16. Oracle Corporation. *SQL*Forms Documentation Addendum. Version 2.3* 1988.
17. Oracle Corporation. *SQL*Forms Operator's Guide Version. 2.3.* 1987.
18. Oracle Corporation. *SQL Language Reference Manual. Version 6.0.* 1989.
19. Oracle Corporation. *SQL*Menu User's Guide. Version 4.0.* 1987.
20. Oracle Corporation. *SQL*PLUS User's Guide and Reference. Version 3.0.* 1989.
21. Ralph B. Bisland, JR *Database Management: Developing Application Systems using Oracle.* Englewood Cliffs, NJ: Prentice-Hall Inc., 1989.
22. Roger S. Pressman. *Software Engineering: A Practitioner's Approach (Second Edition).* New York: McGraw-Hill Publishing Company, 1987.
23. Toby J. Teorey. *Database Modelling and Design: The Entity-Relationship Approach.* San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1990.

24. Tom DeMarco. *Structured Analysis and System Specification.* Englewood Cliffs, NJ: Prectice-Hall Inc., 1979.

25. Wojtkowski W. Gregory and Wojtkowski, Wita. *Applications Software Programming with Fourth-Generation Languages.* Boston: Boyd and Fraster Publishing Company, 1986.

Vita

Captain Yeong-Lae Kwak was born on October 18, 1962, in Geumsan, Korea. He graduated from Kumoh Technical High School in Gumi, in 1981. He entered the Air Force Academy in Seoul, in 1981, where he received Bachelor of Science degree in Electronic Engineering. Upon graduation he was assigned as a second lieutenant of the Air Force. In 1985, he completed the Elementary Computer Course which was offered by Education Command for the officers who are assigned as computer engineers. Also, in 1986, he completed the Software Development Education Course which was offered by Korean Institute of Defense Analysis for the computer engineers of government. He served three years as a computer engineer in the 8th and 15th Wing Computer Center. In 1988, he was assigned to the Headquarter of the Air Force where he served as the Software Development Officer. He entered the School of Engineering, Air Force Institute of Technology of United States, in June, 1990.

Permanent address: 471 Deogchun Namil
Gumsan Chungnam
South Korea

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1992	Master's Thesis

4. TITLE AND SUBTITLE
Development of a Flight Information System using the Structured Method

5. FUNDING NUMBERS

6. AUTHOR(S)
Yeong-lae Kwak, Capt, ROKAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/92M-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstract

This thesis documents the development of a database system for the *Flight Information System* (FIS) of the Korean Air Force. The scope of the FIS is too large to be covered by this thesis. Thus, this thesis covers only the core part of the FIS due to the limitation of time and man-power.

This thesis uses the structured method. Structured analysis and structured design techniques are mainly used two techniques.

This thesis focused not only the development of the FIS but also the application of the software development method, the structured method, and its tools such as DFD, DD, ERD, and so on. Also, the use of ORACLE was a important part of this thesis too.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Database, Oracle, Structured Method	94
	16. PRICE CODE

17 SECURITY CLASSIFICATION OF REPORT	18 SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1 Agency Use Only *(Leave blank)*

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s) Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number *(If known)*

Block 11. Supplementary Notes Enter information not included elsewhere such as: Prepared in cooperation with ; Trans. of ; To be published in . When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank
- **NTIS** - Leave blank

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages Enter the total number of pages

Block 16. Price Code Enter appropriate price code *(NTIS only)*

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.